HTTP Working Group
INTERNET-DRAFT
<draft-ietf-http-v10-spec-01.ps>
Expires February 3, 1996

T. Berners-Lee, MIT/W3C
R. Fielding, UC Irvine
H. Nielsen, MIT/W3C
August 3, 1995

Hypertext Transfer Protocol — HTTP/1.0

Status of this Memo

This document is an Internet-Draft. 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".

To learn the current status of any Internet-Draft, please check the "`1id-abstracts.txt`" listing contained in the Internet-Drafts Shadow Directories on `ftp.is.co.za` (Africa), `nic.nordu.net` (Europe), `munnari.oz.au` (Pacific Rim), `ds.internic.net` (US East Coast), or `ftp.isi.edu` (US West Coast).

Distribution of this document is unlimited. Please send comments to the HTTP working group at <`http-wg@cuckoo.hpl.hp.com`>. Discussions of the working group are archived at <`URL:http://www.ics.uci.edu/pub/ietf/http/`>. General discussions about HTTP and the applications which use HTTP should take place on the <`www-talk@w3.org`> mailing list.

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems. It is a generic, stateless, object-oriented protocol which can be used for many tasks, such as name servers and distributed object management systems, through extension of its request methods (commands). A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred.

HTTP has been in use by the World-Wide Web global information initiative since 1990. This specification reflects preferred usage of the protocol referred to as "HTTP/1.0".

Table of Contents

1. Introduction....5
   - 1.1 Purpose....5
   - 1.2 Overall Operation....5
   - 1.3 Terminology....6
2. Notational Conventions and Generic Grammar....7
   - 2.1 Augmented BNF....7
   - 2.2 Basic Rules....8
3. Protocol Parameters....10
   - 3.1 HTTP Version....10
   - 3.2 Uniform Resource Identifiers....11
     - 3.2.1 General Syntax....11
     - 3.2.2 http URL....12
   - 3.3 Date/Time Formats....12
     - 3.3.1 Full Date....12
     - 3.3.2 Delta Seconds....13
   - 3.4 Media Types....13
     - 3.4.1 Canonicalization and Text Defaults....14
     - 3.4.2 Multipart Types....15
   - 3.5 Character Set Encodings....15
   - 3.6 Encoding Mechanisms....16
   - 3.7 Transfer Encodings....17
   - 3.8 Language Tags....17
   - 3.9 Quality Values....18
   - 3.10 Product Tokens....18
4. HTTP Message....19
   - 4.1 Message Types....19
   - 4.2 Message Headers....19
   - 4.3 General Message Header Fields....20
5. Request....21
   - 5.1 Request-Line....21
   - 5.2 Method....21
     - 5.2.1 GET....22
     - 5.2.2 HEAD....22
     - 5.2.3 POST....22
     - 5.2.4 PUT....23
     - 5.2.5 DELETE....24
     - 5.2.6 LINK....24
     - 5.2.7 UNLINK....24
   - 5.3 Request-URI....25
   - 5.4 Request Header Fields....25

6. Response 26
6.1 Status-Line 26
6.2 Status Codes and Reason Phrases 26
6.2.1 Informational 1xx 28
6.2.2 Successful 2xx 28
6.2.3 Redirection 3xx 29
6.2.4 Client Error 4xx 30
6.2.5 Server Errors 5xx 33
6.3 Response Header Fields 33
7. Entity 34
7.1 Entity Header Fields 34
7.2 Entity Body 34
7.2.1 Type 34
7.2.2 Length 35
8. Header Field Definitions 36
8.1 Accept 36
8.2 Accept-Charset 38
8.3 Accept-Encoding 38
8.4 Accept-Language 38
8.5 Allow 39
8.6 Authorization 39
8.7 Content-Encoding 40
8.8 Content-Language 40
8.9 Content-Length 41
8.10 Content-Transfer-Encoding 41
8.11 Content-Type 41
8.12 Date 42
8.13 Expires 42
8.14 Forwarded 43
8.15 From 43
8.16 If-Modified-Since 44
8.17 Last-Modified 44
8.18 Link 45
8.19 Location 45
8.20 MIME-Version 46
8.21 Orig-URI 46
8.22 Pragma 46
8.23 Public 47
8.24 Referer 48
8.25 Retry-After 48
8.26 Server 48
8.27 Title 49
8.28 URI 49
8.29 User-Agent 50
8.30 WWW-Authenticate 50
9. Content Negotiation 51

10. Access Authentication 53
10.1 Basic Authentication Scheme 54
11. Security Considerations 55
11.1 Authentication of Clients 55
11.2 Idempotent Methods 55
11.3 Abuse of Server Log Information 55
11.4 Transfer of Sensitive Information 55
12. Acknowledgments 57
13. References 58
14. Authors' Addresses 59
Appendix A. Internet Media Type message/http 60
Appendix B. Tolerant Applications 60
Appendix C. Relationship to MIME 61
C.1 Conversion to Canonical Form 61
C.1.1 Representation of Line Breaks 61
C.1.2 Default Character Set Encoding 61
C.2 Default Content-Transfer-Encoding 62
C.3 Introduction of Content-Encoding 62

1. Introduction

1.1 Purpose

The Hypertext Transfer Protocol (HTTP) is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World-Wide Web global information initiative since 1990. This specification reflects preferred usage of the protocol referred to as "HTTP/1.0". This specification does not necessarily reflect the "current practice" of any single HTTP server or client implementation. It does, however, seek to remain compatible with existing implementations wherever possible, and is the reference for future implementations of HTTP/1.0.

Practical information systems require more functionality than simple retrieval, including search, front-end update, and annotation. HTTP/1.0 allows an open-ended set of methods to be used to indicate the purpose of a request. It builds on the discipline of reference provided by the Uniform Resource Identifier (URI) [3], as a location (URL) [5] or name (URN) [18], for indicating the resource on which a method is to be applied. Messages are passed in a format similar to that used by Internet Mail [8] and the Multipurpose Internet Mail Extensions (MIME) [6].

HTTP/1.0 is also used for communication between user agents and various gateways, allowing hypermedia access to existing Internet protocols like SMTP [14], NNTP [12], FTP [16], Gopher [2], and WAIS [9]. HTTP/1.0 is designed to allow such gateways, via proxy servers, without any loss of the data conveyed by those earlier protocols.

1.2 Overall Operation

The HTTP protocol is based on a request/response paradigm. A requesting program (termed a client) establishes a connection with a receiving program (termed a server) and sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content. The server responds with a status line, including its protocol version and a success or error code, followed by a MIME-like message containing server information, entity metainformation, and possible body content. It should be noted that a given program may be capable of being both a client and a server; our use of those terms refers only to the role being performed by the program during a particular connection, rather than to the program's purpose in general.

On the Internet, the communication generally takes place over a TCP/IP connection. The default port is TCP 80 [17], but other ports can be used. This does not preclude the HTTP/1.0 protocol from being implemented on top of any other protocol on the Internet, or on other networks. The mapping of the HTTP/1.0 request and response structures onto the transport data units of the protocol in question is outside the scope of this specification.

For most implementations, the connection is established by the client prior to each request and closed by the server after sending the response. However, this is not a feature of the protocol and is not required by this specification. Both clients and servers must be capable of handling cases where either party closes the connection prematurely, due to user action, automated time-out, or program failure. In any case, the closing of the connection by either or both parties always terminates the current request, regardless of its status.

1.3 Terminology

This specification uses a number of terms to refer to the roles played by participants in, and objects of, the HTTP communication.

connection
A virtual circuit established between two parties for the purpose of communication.

message
A structured sequence of octets transmitted via the connection as the basic component of communication.

request
An HTTP request message (as defined in Section 5).

response
An HTTP response message (as defined in Section 6).

resource
A network data object or service which can be identified by a URI.

entity
A particular representation or rendition of a resource that may be enclosed within a request or response message. An entity consists of metainformation in the form of entity headers and content in the form of an entity body.

client
A program that establishes connections for the purpose of sending requests.

user agent
The client program which is closest to the user and which initiates requests at their behest.

server
A program that accepts connections in order to service requests by sending back responses.

origin server
The server on which a given resource resides or is to be created.

proxy
An intermediary program which acts as both a server and a client for the purpose of forwarding requests. Proxies are often used to act as a portal through a network firewall. A *proxy server* accepts requests from other clients and services them either internally or by passing them, with possible translation, on to other servers. A *caching proxy* is a proxy server with a local cache of server responses — some requested resources can be serviced from the cache rather than from the origin server. Some proxy servers also act as origin servers.

gateway
A proxy which services HTTP requests by translation into protocols other than HTTP. The reply sent from the remote server to the gateway is likewise translated into HTTP before being forwarded to the user agent.

2. Notational Conventions and Generic Grammar

2.1 Augmented BNF

All of the mechanisms specified in this document are described in both prose and an augmented Backus-Naur Form (BNF) similar to that used by RFC 822 [8]. Implementors will need to be familiar with the notation in order to understand this specification. The augmented BNF includes the following constructs:

name = definition

The name of a rule is simply the name itself (without any enclosing "<" and ">") and is separated from its definition by the equal character "=". Whitespace is only significant in that indentation of continuation lines is used to indicate a rule definition that spans more than one line. Certain basic rules are in uppercase, such as SP, LWS, HT, CRLF, DIGIT, ALPHA, etc. Angle brackets are used within definitions whenever their presence will facilitate discerning the use of rule names.

"literal"

Quotation marks surround literal text. Unless stated otherwise, the text is case-insensitive.

rule1 | rule2

Elements separated by a bar ("|") are alternatives, e.g., "yes | no" will accept yes or no.

(rule1 rule2)

Elements enclosed in parentheses are treated as a single element. Thus, "(elem (foo | bar) elem)" allows the token sequences "elem foo elem" and "elem bar elem".

*rule

The character "*" preceding an element indicates repetition. The full form is "<n>*<m>element" indicating at least <n> and at most <m> occurrences of element. Default values are 0 and infinity so that "*(element)" allows any number, including zero; "1*element" requires at least one; and "1*2element" allows one or two.

[rule]

Square brackets enclose optional elements; "[foo bar]" is equivalent to "*1(foo bar)".

N rule

Specific repetition: "<n>(element)" is equivalent to "<n>*<n>(element)"; that is, exactly <n> occurrences of (element). Thus 2DIGIT is a 2-digit number, and 3ALPHA is a string of three alphabetic characters.

#rule

A construct "#" is defined, similar to "*", for defining lists of elements. The full form is "<n>#<m>element" indicating at least <n> and at most <m> elements, each separated by one or more commas (",") and optional linear whitespace (LWS). This makes the usual form of lists very easy; a rule such as "(*LWS element *(*LWS "," *LWS element))" can be shown as "1#element". Wherever this construct is used, null elements are allowed, but do not contribute to the count of elements present. That is, "(element), , (element)" is permitted, but counts as only two elements. Therefore, where at least one element is required, at least one non-null element must be present. Default values are 0 and infinity so that "#(element)" allows any number, including zero; "1#element" requires at least one; and "1#2element" allows one or two.

; comment

A semi-colon, set off some distance to the right of rule text, starts a comment that continues to the end of line. This is a simple way of including useful notes in parallel with the specifications.

implied *LWS

The grammar described by this specification is word-based. Except where noted otherwise, zero or more linear whitespace (LWS) can be included between any two adjacent words (token or quoted-string), and between adjacent tokens and delimiters (tspecials), without changing the interpretation of a field. However, applications should attempt to follow "common form" when generating HTTP constructs, since there exist some implementations that fail to accept anything beyond the common forms.

2.2 Basic Rules

The following rules are used throughout this specification to describe basic parsing constructs. The US-ASCII character set encoding is defined by [19].

```
OCTET          = <any 8-bit sequence of data>
CHAR           = <any US-ASCII character (octets 0 - 127)>
UPALPHA        = <any US-ASCII uppercase letter "A".."Z">
LOALPHA        = <any US-ASCII lowercase letter "a".."z">
ALPHA          = UPALPHA | LOALPHA
DIGIT          = <any US-ASCII digit "0".."9">
CTL            = <any US-ASCII control character
                 (octets 0 - 31) and DEL (127)>
CR             = <US-ASCII CR, carriage return (13)>
LF             = <US-ASCII LF, linefeed (10)>
SP             = <US-ASCII SP, space (32)>
HT             = <US-ASCII HT, horizontal-tab (9)>
<">            = <US-ASCII double-quote mark (34)>
```

HTTP/1.0 defines the octet sequence CR LF as the end-of-line marker for all protocol elements except the Entity-Body (see Appendix B for tolerant applications). The end-of-line marker within an Entity-Body is defined by its associated media type, as described in Section 3.4.

```
CRLF           = CR LF
```

HTTP/1.0 headers can be folded onto multiple lines if the continuation lines begin with linear whitespace characters. All linear whitespace, including folding, has the same semantics as SP.

```
LWS            = [CRLF] 1*( SP | HT )
```

Many HTTP/1.0 header field values consist of words separated by LWS or special characters. These special characters must be in a quoted string to be used within a parameter value.

```
word           = token | quoted-string

token          = 1*<any CHAR except CTLs or tspecials>

tspecials      = "(" | ")" | "<" | ">" | "@"
               | "," | ";" | ":" | "\" | <">
               | "/" | "[" | "]" | "?" | "="
               | SP | HT
```

Comments can be included in HTTP header fields by surrounding the comment text with parentheses.

```
comment        = "(" *( ctext | comment ) ")"
ctext          = <any text excluding "(" and ")">
```

Note: *Use of comments within HTTP headers is generally discouraged, since they are rarely seen by human eyes and hence only increase network traffic. However, they may be useful for messages posted or retrieved via NNTP and SMTP gateways.*

A string of text is parsed as a single word if it is quoted using double-quote marks.

```
quoted-string  = ( <"> *(qdtext) <"> )
qdtext         = <any CHAR except <"> and CTLs,
                 but including LWS>
```

The backslash character ("\") may be used as a single-character quoting mechanism only within quoted-string and comment constructs.

```
quoted-pair    = "\" CHAR
```

When left unquoted and not within a comment, HTTP uses angle brackets to delimit machine-processable addresses; any LWS inside the angle brackets should be ignored.

```
addr-string    = ( "<" *(qatext) ">" )
qatext         = <any CHAR except "<", ">", and CTLs,
                 but including LWS>
```

The text rule is only used for descriptive field contents and values that are not intended to be interpreted by the message parser. Words of *text may contain octets from character set encodings other than US-ASCII only when encoded according to the rules of RFC 1522 [13].

```
text           = <any OCTET except CTLs,
                 but including LWS>
```

Recipients of header field text containing octets outside the US-ASCII character set encoding may assume that they are ISO-8859-1 characters if there is no other encoding indicated by an RFC 1522 mechanism.

3. Protocol Parameters

3.1 HTTP Version

HTTP uses a "<major>.<minor>" numbering scheme to indicate versions of the protocol. The protocol versioning policy is intended to allow the sender to indicate the format of a message and its capacity for understanding further HTTP communication, rather than the features obtained via that communication. No change is made to the version number for the addition of message components which do not affect communication behavior or which only add to extensible field values. The <minor> number is incremented when the changes made to the protocol add features which do not change the general message parsing algorithm, but which may add to the message semantics and imply additional capabilities of the sender. The <major> number is incremented when the format of a message within the protocol is changed.

The version of an HTTP message is indicated by an HTTP-Version field in the first line of the message. If the protocol version is not specified, the recipient must assume that the message is in the simple HTTP/0.9 format.

```
HTTP-Version      = "HTTP" "/" 1*DIGIT "." 1*DIGIT
```

Note that the major and minor numbers should be treated as separate integers and that each may be incremented higher than a single digit. Thus, HTTP/2.4 is a lower version than HTTP/2.13, which in turn is lower than HTTP/12.3. Leading zeros should be ignored by recipients and never generated by senders.

This document defines both the 0.9 and 1.0 versions of the HTTP protocol. Applications sending Full-Request or Full-Response messages, as defined by this specification, must include an HTTP-Version of "`HTTP/1.0`".

HTTP servers must be able to recognize the format of the Request-Line for all lower-version requests, to understand any valid request in the format of the immediately-prior major version (<major-1>), to understand any valid request in the format of their own native major version (<major>) with the same or lower minor version, and to respond appropriately with a message within the same <major> protocol version used by the client, even when the response is simply an error message.

HTTP clients must be able to recognize the format of the Status-Line for all lower-version responses, to understand any valid response in the format of the immediately-prior major version (<major-1>), and to understand any valid response in the format of their own native major version (<major>) with the same or lower minor version. The following hypothetical example illustrates the required behavior.

- A valid HTTP/3.5 request is received and the server's native protocol version is
  - Less than 3.0: it should attempt to understand the request and respond (possibly with an error) in its native format;
  - Major number of 3: It should understand the request and respond in its native format;
  - Major number of 4: It should understand the request and respond with a version 3 message;
  - Major number higher than 4: It should attempt to understand the request and respond (possibly with an error) with a version 3 message;

- User agent receives a response to an HTTP/3.5 request, and the response version is
  - Less than 2.0: It should attempt to understand the response and unobtrusively warn the user of the version mismatch;
  - 2.0--3.4: It should understand the response and be aware that its request may not have been fully understood by the server;
  - 3.5 or higher 3: It should understand the response and can assume that the server understood all aspects of the request if the response does not indicate an error;
  - 4.0 or higher: It should attempt to understand the response and unobtrusively warn the user of the version mismatch.

Proxies must be careful in forwarding requests that are received in a format different than that of the proxy's native version. Since the protocol version indicates the protocol capability of the sender, a proxy must never send a message with a version indicator which is greater than its native version; if a higher version request is received, the proxy must either downgrade the request version or respond with an error. Requests with a version lower than that of the proxy's native format may be upgraded by the proxy before being forwarded; the proxy's response to that request must follow the normal server requirements.

3.2 Uniform Resource Identifiers

URIs have been known by many names: WWW addresses, Universal Document Identifiers, Universal Resource Identifiers [3], and finally the combination of Uniform Resource Locators (URL) [5] and Names (URN) [18]. As far as HTTP is concerned, Uniform Resource Identifiers are simply formatted strings which identify—via name, location, or any other characteristic—a network resource.

3.2.1 General Syntax

URIs in HTTP/1.0 can be represented in absolute form or relative to some known base URI [10], depending upon the context of their use. The two forms are differentiated by the fact that absolute URIs always begin with a scheme name followed by a colon.

```
URI            = ( absoluteURI | relativeURI ) [ "#" fragment ]

absoluteURI    = scheme ":" *( uchar | reserved )

relativeURI    = net_path | abs_path | rel_path

net_path       = "//" net_loc [ abs_path ]
abs_path       = "/" rel_path
rel_path       = [ path ] [ ";" params ] [ "?" query ]

path           = fsegment *( "/" segment )
fsegment       = 1*pchar
segment        = *pchar

params         = param *( ";" param )
param          = *( pchar | "/" )

scheme         = 1*( ALPHA | DIGIT | "+" | "-" | "." )
net_loc        = *( pchar | ";" | "?" )
query          = *( uchar | reserved )
fragment       = *( uchar | reserved )

pchar          = uchar | ":" | "@" | "&" | "="
uchar          = unreserved | escape
unreserved     = ALPHA | DIGIT | safe | extra | national
```

```
escape         = "%" hex hex
hex            = "A" | "B" | "C" | "D" | "E" | "F"
               |  "a" | "b" | "c" | "d" | "e" | "f" | DIGIT
reserved       = ";" | "/" | "?" | ":" | "@" | "&" | "="
safe           = "$" | "-" | "_" | "." | "+"
extra          = "!" | "*" | "'" | "(" | ")" | ","
national       = <any OCTET excluding CTLs, SP,
                  ALPHA, DIGIT, reserved, safe, and extra>
```

For more information on URL syntax and semantics, see RFC 1738 [5] and RFC 1808 [10]. The BNF above includes characters—all those marked as national—not allowed in valid URLs as specified by RFC 1738, since HTTP servers are not restricted in the set of unreserved characters allowed to represent the rel_path part of addresses. In fact, the only real requirement for HTTP is that the URI not contain any LWS; any other invalid URI can be identified and rejected by the server.

3.2.2 http URL

The "http" scheme is used to locate network resources via the HTTP protocol. This section defines the scheme-specific syntax and semantics for http URLs.

```
http_URL       = "http:" "//" host [ ":" port ] abs_path

host           = <FQDN or IP address, as defined in RFC 1738>
port           = *DIGIT
```

If the port is empty or not given, port 80 is assumed. The semantics are that the identified resource is located at the server listening for TCP connections on that port of that host, and the Request-URI for the resource is abs_path. If the abs_path is not present in the URL, it must be given as "/" when used as a Request-URI.

The canonical form for "http" URLs is obtained by converting any UPALPHA characters in host to their LOALPHA equivalent (hostnames are case-insensitive), eliding the [":" port] if the port is 80, and replacing an empty abs_path with "/".

3.3 Date/Time Formats

3.3.1 Full Date

HTTP/1.0 applications have historically allowed three different formats for the representation of date/time stamps:

```
Sun, 06 Nov 1994 08:49:37 GMT    ; RFC 822, updated by RFC 1123
Sunday, 06-Nov-94 08:49:37 GMT   ; RFC 850, obsoleted by RFC 1036
Sun Nov  6 08:49:37 1994         ; ANSI C's asctime() format
```

The first format is preferred as an Internet standard and represents a fixed-length subset of that defined by RFC 1123 [7] (an update to RFC 822 [8]). The second format is in common use, but is based on the obsolete RFC 850 [11] date format and lacks a four-digit year. HTTP/1.0 clients and servers must accept all three formats, though they must never generate the third (asctime) format. Future clients and servers must only generate the RFC 1123 format for representing date/time stamps in HTTP/1.0 requests and responses.

> ***Note:*** *Recipients of date values are encouraged to be robust in accepting date values that may have been generated by non-HTTP applications, as is sometimes the case when retrieving or posting messages via gateways to SMTP or NNTP.*

All HTTP/1.0 date/time stamps must be represented in Universal Time (UT), also known as Greenwich Mean Time (GMT), without exception. This is indicated in the first two formats by the inclusion of “GMT” as the three-letter abbreviation for time zone, and should be assumed when reading the asctime format.

```
HTTP-date      = rfc1123-date | rfc850-date | asctime-date

rfc1123-date   = wkday "," SP date1 SP time SP "GMT"
rfc850-date    = weekday "," SP date2 SP time SP "GMT"
asctime-date   = wkday SP date3 SP time SP 4DIGIT

date1          = 2DIGIT SP month SP 4DIGIT
                 ; day month year (e.g., 02 Jun 1982)
date2          = 2DIGIT "-" month "-" 2DIGIT
                 ; day-month-year (e.g., 02-Jun-82)
date3          = month SP ( 2DIGIT | ( SP 1DIGIT ))
                 ; month day (e.g., Jun  2)

time           = 2DIGIT ":" 2DIGIT ":" 2DIGIT
                 ; 00:00:00 - 23:59:59

wkday          = "Mon" | "Tue" | "Wed"
               | "Thu" | "Fri" | "Sat" | "Sun"

weekday        = "Monday" | "Tuesday" | "Wednesday"
               | "Thursday" | "Friday" | "Saturday" | "Sunday"

month          = "Jan" | "Feb" | "Mar" | "Apr"
               | "May" | "Jun" | "Jul" | "Aug"
               | "Sep" | "Oct" | "Nov" | "Dec"
```

Comments and/or extra LWS are not permitted inside an HTTP-date value generated by a conformant application.

> ***Note:*** *HTTP/1.0 requirements for the date/time stamp format apply only to their usage within the protocol stream. Clients and servers are not required to use these formats for user presentation, request logging, etc.*

3.3.2 Delta Seconds

Some HTTP header fields allow a time value to be specified as an integer number of seconds, represented in decimal, after the time that the message was received. This format should only be used to represent short time periods or periods that cannot start until receipt of the message.

```
delta-seconds  = 1*DIGIT
```

3.4 Media Types

HTTP uses Internet Media Types [15] (formerly referred to as MIME Content-Types [6]) in order to provide open and extensible data typing and type negotiation. For mail applications, where there is no type negotiation between sender and receiver, it is reasonable to put strict limits on the set of allowed media types. With HTTP, however, user agents can identify acceptable media types as part of the connection, and thus are allowed more freedom in the use of non-registered types. The following grammar for media types is a superset of that for MIME because it does not restrict itself to the official IANA and x-token types.

```
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 not case-sensitive. Parameter values may or may not be case-sensitive, depending on the semantics of the parameter name. LWS should not be generated between the type and subtype, nor between an attribute and its value.

If a given media-type value has been registered by the IANA, any use of that value must be indicative of the registered data format. Although HTTP allows the use of non-registered media types, such usage must not conflict with the IANA registry. Data providers are strongly encouraged to register their media types with IANA via the procedures outlined in RFC 1590 [15].

All media-type's registered by IANA must be preferred over extension tokens. However, HTTP does not limit conforming applications to the use of officially registered media types, nor does it encourage the use of an "x-" prefix for unofficial types outside of explicitly short experimental use between consenting applications.

3.4.1 Canonicalization and Text Defaults

Media types are registered in a canonical form. In general, entity bodies transferred via HTTP must be represented in the appropriate canonical form prior to transmission. If the body has been encoded via a Content-Encoding and/or Content-Transfer-Encoding, the data must be in canonical form prior to that encoding. However, HTTP modifies the canonical form requirements for media of primary type "text" and for "application" types consisting of text-like records.

HTTP redefines the canonical form of text media to allow multiple octet sequences to indicate a text line break. In addition to the preferred form of CRLF, HTTP applications must accept a bare CR or LF alone as representing a single line break in text media. Furthermore, if the text media is represented in a character set encoding which does not use octets 13 and 10 for CR and LF respectively, as is the case for some multi-byte character set encodings, HTTP allows the use of whatever octet sequence(s) is defined by that character set encoding to represent the equivalent of CRLF, bare CR, and bare LF. It is assumed that any recipient capable of using such a character set encoding will know the appropriate octet sequence for representing line breaks within that character set encoding.

> ***Note:*** *This interpretation of line breaks applies only to the contents of an Entity-Body and only after any Content-Transfer-Encoding and/or Content-Encoding has been removed. All other HTTP constructs use CRLF exclusively to indicate a line break. Encoding mechanisms define their own line break requirements.*

A recipient of an HTTP text entity should translate the received entity line breaks to the local line break conventions before saving the entity external to the application and its cache; whether this translation takes place immediately upon receipt of the entity, or only when prompted by the user, is entirely up to the individual application.

HTTP also redefines the default character set encoding for text media in an entity body. If a textual media type defines a charset parameter with a registered default value of "US-ASCII", HTTP changes the default to be "ISO-8859-1". Since the ISO-8859-1 [20] character set encoding is a superset of US-ASCII [19], this has no effect upon the interpretation of entity bodies which only contain octets within the US-ASCII set (0 - 127). The presence of a charset parameter value in a Content-Type header field overrides the default.

It is recommended but not required that the character set encoding of an entity body be labelled as the lowest common denominator of the character codes used within a document, with the exception that no label is preferred over the labels US-ASCII or ISO-8859-1.

3.4.2 Multipart Types

MIME provides for a number of "multipart" types — encapsulations of several entities within a single message's Entity-Body. The multipart types registered by IANA [17] do not have any special meaning for HTTP/1.0, though user agents may need to understand each type in order to correctly interpret the purpose of each body-part. Ideally, an HTTP user agent should follow the same or similar behavior as a MIME user agent does upon receipt of a multipart type.

As in MIME [6], all multipart types share a common syntax 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 MIME, multipart body-parts may contain HTTP header fields which are significant to the meaning of that part.

A URI-header field (Section 8.28) should be included in the body-part for each enclosed entity that can be identified by a URI.

3.5 Character Set Encodings

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 encodings, 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.

However, since this is more commonly referred to as a character encoding, this document will refer to them as character set encodings. Character set encodings are identified by case-insensitive tokens. The complete set of tokens are defined by the IANA Character Set registry [17]. However, because that registry does not define a single, consistent token for each character set encoding, we define here the preferred names for those character set encodings most likely to be used with HTTP entities. This set of charset values includes those registered by RFC 1521 [6] — the US-ASCII [19] and ISO-8859 [20] character set encodings — and other names specifically recommended for use within MIME charset parameters.

```
charset  = "US-ASCII"
         | "ISO-8859-1" | "ISO-8859-2" | "ISO-8859-3"
         | "ISO-8859-4" | "ISO-8859-5" | "ISO-8859-6"
         | "ISO-8859-7" | "ISO-8859-8" | "ISO-8859-9"
         | "ISO-2022-JP" | "ISO-2022-JP-2" | "ISO-2022-KR"
         | "UNICODE-1-1" | "UNICODE-1-1-UTF-7" | "UNICODE-1-1-UTF-8"
         | 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 [17] must represent the character set encoding defined by that registry. Applications are encouraged, but not required, to limit their use of character set encodings to those defined by the IANA registry.

3.6 Encoding Mechanisms

Encoding mechanism values are used to indicate an encoding transformation that has been or can be applied to a resource. Encoding mechanisms are primarily used to allow a document to be compressed or encrypted without losing the identity of its underlying media type. Typically, the resource is stored in this encoding and only decoded before rendering or analogous usage.

```
encoding-mechanism       = "gzip" | "compress" | token
```

Note: *For historical reasons, HTTP/1.0 applications should consider "x-gzip" and "x-compress" to be equivalent to "gzip" and "compress", respectively.*

All encoding-mechanism values are case-insensitive. HTTP/1.0 uses encoding-mechanism values in the Accept-Encoding (Section 8.3) and Content-Encoding (Section 8.7) header fields. Although the value describes the encoding-mechanism, what is more important is that it indicates what decoding mechanism will be required to remove the encoding. Note that a single program may be capable of decoding multiple encoding-mechanism formats. Two values are defined by this specification:

gzip An encoding format produced by the file compression program "gzip" (GNU zip) developed by Jean-loup Gailly. This format is typically a Lempel-Ziv coding (LZ77) with a 32 bit CRC. Gzip is available from the GNU project at <URL:ftp://prep.ai.mit.edu/pub/gnu/>.

compress The encoding format produced by the file compression program "compress". This format is an adaptive Lempel-Ziv-Welch coding (LZW).

Note: *Use of program names for the identification of encoding formats is not desirable and should be discouraged for future encodings. Their use here is representative of historical practice, not good design.*

3.7 Transfer Encodings

Transfer encoding values are used to indicate an encoding transformation that has been, can be, or may need to be applied to an Entity-Body in order to ensure safe transport through the network. Current transfer encodings are only used with entities destined for or retrieved from MIME-conformant systems, and thus will rarely occur in an HTTP/1.0 message. This differs from an encoding-mechanism in that the transfer encoding is a property of the message, not of the original resource.

```
transfer-encoding        = "binary" | "8bit" | "7bit"
                         | "quoted-printable" | "base64"
                         | token
```

All transfer-encoding values are case-insensitive. HTTP/1.0 may use transfer-encoding values in the Content-Transfer-Encoding (Section 8.10) header field.

> ***Note:*** *Transfer encodings were designed for MIME with the assumption of their being used only within the context of Internet mail and SMTP. "Safe transport" has a different focus for an 8bit-clean transfer protocol. In HTTP, the only unsafe characteristic of message bodies is the difficulty in determining the exact body length (Section 7.2.2).*

The values "7bit", "8bit", and "binary" are used to indicate that no transfer encoding has been performed. Instead, they describe the sort of encoding that might be needed for transmission through an unsafe transport system. Binary indicates that the body may contain any set of octets. 8bit adds the restrictions that CR and LF characters only occur as part of CRLF line separators, all lines are short (less than 1000 octets), and no NULs (octet 0) are present. 7bit adds a further restriction that all octets are 7-bit US-ASCII characters.

The "quoted-printable" and "base64" values indicate that the associated encoding (as defined in MIME [6]) has been applied to the body. These encodings consist entirely of 7-bit US-ASCII characters.

3.8 Language Tags

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. The HTTP/1.0 protocol uses language tags within the Accept-Language, Content-Language, and URI-header fields.

The syntax and registry of HTTP language tags is the same as that defined by RFC 1766 [1]. 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
```

Whitespace is not allowed within the tag and all tags are not case-sensitive. The namespace 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.

In the context of the Accept-Language header (Section 8.4), a language tag is not to be interpreted as a single token, as per RFC 1766, but as a hierarchy. A server should consider that it has a match when a language tag received in an Accept-Language header matches the initial portion of the language tag of a document. An exact match should be preferred. This interpretation allows a browser to send, for example:

```
Accept-Language: en-US, en; ql=0.95
```

when the intent is to access, in order of preference, documents in US-English ("en-US"), 'plain' or 'international' English ("en"), and any other variant of English (initial "en-").

Note: *Using the language tag as a hierarchy does not imply that all languages with a common prefix will be understood by those fluent in one or more of those languages; it simply allows the user to request this commonality when it is true for that user.*

3.9 Quality Values

HTTP content negotiation (Section 9) uses short "floating point" numbers to indicate the relative importance ("weight") of various negotiable parameters. The calculated weights are normalized to a real number in the range 0 through 1, where 0 is the minimum and 1 the maximum value. In order to discourage misuse of this feature, HTTP/1.0 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 ] )
                | ( "." 0*3DIGIT )
                | ( "1" [ "." 0*3("0") ] )
```

"Quality values" is a slight misnomer, since these values actually measure relative degradation in perceived quality. Thus, a value of "0.8" represents a 20% degradation from the optimum rather than a statement of 80% quality.

3.10 Product Tokens

Product tokens are used to allow communicating applications to identify themselves via a simple product token, with an optional slash and version designator. Most fields using product tokens also allow subproducts which form a significant part of the application to be listed, separated by whitespace. By convention, the products are listed in order of their significance for identifying the application.

```
product         = token ["/" product-version]
product-version = token
```

Examples:

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

Server: Apache/0.8.4
```

Product tokens should be short and to the point — use of them for advertizing or other non-essential information is explicitly forbidden. Although any token character may appear in a product-version, this token should only be used for a version identifier (i.e., successive versions of the same product should only differ in the product-version portion of the product value).

4. HTTP Message

4.1 Message Types

HTTP messages consist of requests from client to server and responses from server to client.

```
HTTP-message   = Simple-Request           ; HTTP/0.9 messages
               | Simple-Response
               | Full-Request             ; HTTP/1.0 messages
               | Full-Response
```

Full-Request and Full-Response use the generic message format of RFC 822 [8] for transferring entities. Both messages may include optional header fields (a.k.a. "headers") and an entity body. The entity body is separated from the headers by a null line (i.e., a line with nothing preceding the CRLF).

```
Full-Request   = Request-Line             ; Section 5.1
                 *( General-Header        ; Section 4.3
                  | Request-Header        ; Section 5.4
                  | Entity-Header )       ; Section 7.1
                 CRLF
                 [ Entity-Body ]          ; Section 7.2

Full-Response  = Status-Line              ; Section 6.1
                 *( General-Header        ; Section 4.3
                  | Response-Header       ; Section 6.3
                  | Entity-Header )       ; Section 7.1
                 CRLF
                 [ Entity-Body ]          ; Section 7.2
```

Simple-Request and Simple-Response do not allow the use of any header information and are limited to a single request method (GET).

```
Simple-Request  = "GET" SP Request-URI CRLF

Simple-Response = [ Entity-Body ]
```

Use of the Simple-Request format is discouraged because it prevents the client from using content negotiation and the server from identifying the media type of the returned entity.

4.2 Message Headers

HTTP header fields, which include General-Header (Section 4.3), Request-Header (Section 5.4), Response-Header (Section 6.3), and Entity-Header (Section 7.1) fields, follow the same generic format as that given in Section 3.1 of RFC 822 [8]. Each header field consists of a name followed by a colon (":") and the field value. Field names are never case-sensitive. The field value may be preceded by any amount of LWS, though a single SP is preferred. Header fields can be extended over multiple lines by preceding each extra line with at least one LWS.

```
HTTP-header    = field-name ":" [ field-value ] CRLF

field-name     = 1*<any CHAR, excluding CTLs, SP, and ":">
field-value    = *( field-content | comment | LWS )

field-content  = <the OCTETs making up the field-value
                 and consisting of either *text or combinations
                 of token, tspecials, and quoted-string>
```

The order in which header fields are received is not significant. However, it is "good practice" to send General-Header fields first, followed by Request-Header or Response-Header fields prior to the Entity-Header fields.

Multiple HTTP-header fields with the same field-name may be present in a message if and only if the entire field-value for that header field is defined as a comma-separated list [i.e., #(values)]. It must be possible to combine the multiple header fields into one "field-name: field-value" pair, without changing the semantics of the message, by appending each subsequent field-value to the first, each separated by a comma.

4.3 General Message Header Fields

There are a few header fields which have general applicability for both request and response messages, but which do not apply to the communicating parties or the content being transferred. Although none of the General-Header fields are required, they are all strongly recommended where their use is appropriate, and should be understood by all future HTTP/1.0 clients and servers. These headers apply only to the message being transmitted.

```
General-Header  = Date                 ; Section 8.12
                | Forwarded            ; Section 8.14
                | MIME-Version         ; Section 8.20
                | Pragma               ; Section 8.22
```

General header field names can be extended only via a change in the protocol version. Unknown header fields are treated as Entity-Header fields.

5. Request

A request message from a client to a server includes, within the first line of that message, the method to be applied to the resource requested, the identifier of the resource, and the protocol version in use. For backwards compatibility with the more limited HTTP/0.9 protocol, there are two valid formats for an HTTP request:

```
Request         = Simple-Request | Full-Request

Simple-Request  = "GET" SP Request-URI CRLF

Full-Request    = Request-Line              ; Section 5.1
                  *( General-Header         ; Section 4.3
                   | Request-Header         ; Section 5.4
                   | Entity-Header )        ; Section 7.1
                  CRLF
                  [ Entity-Body ]           ; Section 7.2
```

If an HTTP/1.0 server receives a Simple-Request, it must respond with an HTTP/0.9 Simple-Response. An HTTP/1.0 client capable of receiving a Full-Response should never generate a Simple-Request.

5.1 Request-Line

The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. The elements are separated by SP characters. No CR or LF are allowed except in the final CRLF sequence.

```
Request-Line    = Method SP Request-URI SP HTTP-Version CRLF
```

Note that the difference between a Simple-Request and the Request-Line of a Full-Request is the presence of the HTTP-Version field and the availability of methods other than "GET".

5.2 Method

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

```
Method          = "GET" | "HEAD" | "PUT" | "POST"
                | "DELETE" | "LINK" | "UNLINK"
                | extension-method

extension-method = token
```

The list of methods acceptable by a specific resource can be specified in an “Allow” Entity-Header (Section 8.5). However, the client is always notified through the return code of the response whether a method is currently allowed on a specific resource, as this can change dynamically. Servers should return the status code “405 Method Not Allowed” if the method is known by the server but not allowed for the requested resource, and “501 Not Implemented” if the method is unknown or not implemented by the server.

The methods GET and HEAD must be supported by all general-purpose servers. Servers which provide Last-Modified dates for resources must also support the conditional GET method.

The set of common methods for HTTP/1.0 is described below. Although this set can be easily expanded, additional methods cannot be assumed to share the same semantics for separately extended clients and servers. In order to maintain compatibility, the semantic definition for extension methods should be registered with the IANA [17].

5.2.1 GET

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

The semantics of the GET method changes to a "conditional GET" if the request message includes an If-Modified-Since header field. A conditional GET method requests that the identified resource be transferred only if it has been modified since the date given by the If-Modified-Since header, as described in Section 8.16. The conditional GET method is intended to reduce network usage by allowing cached entities to be refreshed without requiring multiple requests or transferring unnecessary data.

5.2.2 HEAD

The HEAD method is identical to GET except that the server must not return any Entity-Body in the response. The metainformation contained in the HTTP headers in response to a HEAD request should be identical to the information sent in response to a GET request. This method can be used for obtaining metainformation about the resource identified by the Request-URI without transferring the Entity-Body itself. This method is often used for testing hypertext links for validity, accessibility, and recent modification.

There is no "conditional HEAD" request analogous to the conditional GET. If an If-Modified-Since header field is included with a HEAD request, it should be ignored.

5.2.3 POST

The POST method is used to request that the destination server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. POST is designed to allow a uniform method to cover the following functions:

- Annotation of existing resources;
- Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles;
- Providing a block of data, such as the result of submitting a form [4], to a data-handling process;
- Extending a database through an append operation.

The actual function performed by the POST method is determined by the server and is usually dependent on the Request-URI. The posted entity is subordinate to that URI in the same way that a file is subordinate to a directory containing it, a news article is subordinate to a newsgroup to which it is posted, or a record is subordinate to a database.

The client can suggest a URI for identifying the new resource by including a URI-header field in the request. However, the server should treat that URI as advisory and may store the entity under a different URI or without any URI.

The client may apply relationships between the new resource and other existing resources by including Link header fields, as described in Section 8.18. The server may use the Link information to perform other operations as a result of the new resource being added. For example, lists and indexes might be updated. However, no mandatory operation is imposed on the origin server. The origin server may also generate its own or additional links to other resources.

A successful POST does not require that the entity be created as a resource on the origin server or made accessible for future reference. That is, the action performed by the POST method might not result in a resource that can be identified by a URI. In this case, either 200 (ok) or 204 (no content) is the appropriate response status, depending on whether or not the response includes an entity that describes the result.

If a resource has been created on the origin server, the response should be 201 (created) and contain the allocated URI, all applicable Link header fields, and an entity (preferably of type "text/html") which describes the status of the request and refers to the new resource.

A valid Content-Length is required on all HTTP/1.0 POST requests. An HTTP/1.0 server should respond with a 400 (bad request) message if it cannot determine the length of the request message's content.

5.2.4 PUT

The PUT method requests that the enclosed entity be stored under the supplied Request-URI. If the Request-URI refers to an already existing resource, the enclosed entity should be considered as a modified version of the one residing on the origin server. If the Request-URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI. If a new resource is created, the origin server must inform the user agent via the 201 (created) response. If an existing resource is modified, either the 200 (ok) or 204 (no content) response codes should be sent to indicate successful completion of the request. If the resource could not be created or modified with the Request-URI, an appropriate error response should be given that reflects the nature of the problem.

The fundamental difference between the POST and PUT requests is reflected in the different meaning of the Request-URI. The URI in a POST request identifies the resource that will handle the enclosed entity as an appendage. That resource may be a data-accepting process, a gateway to some other protocol, or a separate entity that accepts annotations. In contrast, the URI in a PUT request identifies the entity enclosed with the request — the user agent knows what URI is intended and the server must not attempt to apply the request to some other resource. If the server desires that the request be applied to a different URI, it must send a 301 (moved permanently) response; the user agent may then make its own decision regarding whether or not to redirect the request.

A single resource may be identified by many different URIs. For example, an article may have a URI for identifying "the current version" which is separate from the URI identifying each particular version. In this case, a PUT request on a general URI may result in several other URIs being defined by the origin server. The user agent should be informed of these URIs via one or more URI header fields in the response. The Location header field should be used to identify the exact location URI if it is different than the Request-URI.

A valid Content-Length is required on all HTTP/1.0 PUT requests. An HTTP/1.0 server should respond with a 400 (bad request) message if it cannot determine the length of the request message's content.

The client can create or modify relationships between the enclosed entity and other existing resources by including Link header fields, as described in Section 8.18. As with POST, the server may use the Link information to perform other operations as a result of the request. However, no mandatory operation is imposed on the origin server. The origin server may generate its own or additional links to other resources.

The actual method for determining how the resource is placed, and what happens to its predecessor, is defined entirely by the origin server. If version control is implemented by the origin server, then Link relationships should be defined by the server to help identify and control revisions to a resource; suggested relationship names include "Derived-From", "Obsoletes", and "Updates".

> ***Note:*** *The model of sending an entire PUT request within a single message, without first checking if the server is willing to accept that data, will break if the server is unwilling to accept the request or desires some form of authentication beforehand. Worse, the client won't be notified of the reason for error if a TCP reset is received prior to reading the response buffer (see note in Section 6.2.4). It should therefore be recognized that HTTP/1.0 PUT and large POST requests will only work reliably if the client's intentions and server's desires are negotiated prior to the request.*

5.2.5 DELETE

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

A successful response should be 200 (ok) if the response includes an entity describing the status, 202 (accepted) if the action has not yet been enacted, or 204 (no content) if the response is OK but does not include an entity.

5.2.6 LINK

The LINK method establishes one or more Link relationships between the existing resource identified by the Request-URI and other existing resources. The difference between LINK and other methods allowing links to be established between resources is that the LINK method does not allow any Entity-Body to be sent in the request and does not result in the creation of new resources.

5.2.7 UNLINK

The UNLINK method removes one or more Link relationships from the existing resource identified by the Request-URI. These relationships may have been established using the LINK method or by any other method supporting the Link header. The removal of a link to a resource does not imply that the resource ceases to exist or becomes inaccessible for future references.

5.3 Request-URI

The Request-URI is a Uniform Resource Identifier (Section 3.2) and identifies the resource upon which to apply the request.

```
Request-URI    = "*" | absoluteURI | abs_path
```

The three options for Request-URI are dependent on the nature of the request. The asterisk "*" means that the request does not apply to a particular resource, but to the server itself, and is only allowed when the Method used does not necessarily apply to a resource. Note that this is not the case for any of the methods defined by this document; however, it may be true of extension methods. One example would be

```
OPTIONS * HTTP/1.0
```

The absoluteURI form is only allowed when the request is being made to a proxy server. The proxy is requested to forward the request and return the response. If the request is idempotent and a response is cached, the proxy may return the cached message if it passes any restrictions in the Pragma and Expires header fields. Note that the proxy may forward the request on to another proxy or directly to the origin server specified by the absoluteURI. In order to avoid request loops, a proxy must be able to recognize all of its server names, including any aliases, local variations, and the numeric IP address. An example Request-Line would be:

```
GET http://www.w3.org/hypertext/WWW/TheProject.html HTTP/1.0
```

The most common form of Request-URI is that used to identify a resource on an origin server. In this case, only the absolute path of the URI (abs_path) is transmitted. For example, a client wishing to retrieve the resource above directly from the origin server would create a TCP connection to port 80 of the host "www.w3.org" and send the line:

```
GET /hypertext/WWW/TheProject.html HTTP/1.0
```

followed by the remainder of the Full-Request. Note that the absolute path cannot be empty; if none is present in the original URI, it must be given as "/" (the server root).

5.4 Request Header Fields

The request header fields allow the client to pass additional information about the request, and about the client itself, to the server. All header fields are optional and conform to the generic HTTP-header syntax.

```
Request-Header  = Accept                    ; Section 8.1
                | Accept-Charset            ; Section 8.2
                | Accept-Encoding           ; Section 8.3
                | Accept-Language           ; Section 8.4
                | Authorization             ; Section 8.6
                | From                      ; Section 8.15
                | If-Modified-Since         ; Section 8.16
                | Orig-URI                  ; Section 8.21
                | Referer                   ; Section 8.24
                | User-Agent                ; Section 8.29
```

Request-Header field names can be extended only via a change in the protocol version. Unknown header fields are treated as Entity-Header fields.

6. Response

After receiving and interpreting a request message, a server responds in the form of an HTTP response message.

```
Response        = Simple-Response | Full-Response

Simple-Response = [ Entity-Body ]

Full-Response   = Status-Line               ; Section 6.1
                  *( General-Header         ; Section 4.3
                  |  Response-Header        ; Section 6.3
                  |  Entity-Header )        ; Section 7.1
                  CRLF
                  [ Entity-Body ]           ; Section 7.2
```

A Simple-Response should only be sent in response to an HTTP/0.9 Simple-Request or if the server only supports the more limited HTTP/0.9 protocol. If a client sends an HTTP/1.0 Full-Request and receives a response that does not begin with a Status-Line, it should assume that the response is a Simple-Response and parse it accordingly. Note that the Simple-Response consists only of the entity body and is terminated by the server closing the connection.

6.1 Status-Line

The first line of a Full-Response message is the Status-Line, consisting of the protocol version followed by a numeric status code and its associated textual phrase, with each element separated by SP characters. No CR or LF is allowed except in the final CRLF sequence.

```
Status-Line     = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

Since a status line always begins with the protocol version and status code

```
"HTTP/" 1*DIGIT "." 1*DIGIT SP 3DIGIT SP
```

(e.g., `"HTTP/1.0 200 "`), the presence of that expression is sufficient to differentiate a Full-Response from a Simple-Response. Although the Simple-Response format may allow such an expression to occur at the beginning of an entity body, and thus cause a misinterpretation of the message if it was given in response to a Full-Request, most HTTP/0.9 servers are limited to responses of type "text/html" and therefore never generate such a response.

6.2 Status Codes and Reason Phrases

The Status-Code element is a 3-digit integer result code of the attempt to understand and satisfy the request. The Reason-Phrase is intended to give a short textual description of the Status-Code. The Status-Code is intended for use by automata and the Reason-Phrase is intended for the human user. The client is not required to examine or display the Reason-Phrase.

The first digit of the Status-Code defines the class of response. The last two digits do not have any categorization role. There are 5 values for the first digit:

- 1xx: Informational - Not used, but reserved for future use
- 2xx: Success - The action was successfully received, understood, and accepted.
- 3xx: Redirection - Further action must be taken in order to complete the request
- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
- 5xx: Server Error - The server failed to fulfill an apparently valid request

The individual values of the numeric status codes defined for HTTP/1.0, and an example set of corresponding Reason-Phrase's, are presented below. The reason phrases listed here are only recommended — they may be replaced by local equivalents without affecting the protocol.

```
Status-Code    = "200"   ; OK
               | "201"   ; Created
               | "202"   ; Accepted
               | "203"   ; Non-Authoritative Information
               | "204"   ; No Content
               | "300"   ; Multiple Choices
               | "301"   ; Moved Permanently
               | "302"   ; Moved Temporarily
               | "303"   ; See Other
               | "304"   ; Not Modified
               | "400"   ; Bad Request
               | "401"   ; Unauthorized
               | "402"   ; Payment Required
               | "403"   ; Forbidden
               | "404"   ; Not Found
               | "405"   ; Method Not Allowed
               | "406"   ; None Acceptable
               | "407"   ; Proxy Authentication Required
               | "408"   ; Request Timeout
               | "409"   ; Conflict
               | "410"   ; Gone
               | "411"   ; Authorization Refused
               | "500"   ; Internal Server Error
               | "501"   ; Not Implemented
               | "502"   ; Bad Gateway
               | "503"   ; Service Unavailable
               | "504"   ; Gateway Timeout
               | extension-code

extension-code = 3DIGIT

Reason-Phrase  = *<text, excluding CR, LF>
```

HTTP status codes are extensible and should be registered with the IANA. HTTP applications are not required to understand the meaning of all registered status codes, though such understanding is obviously desirable. However, applications must understand the class of any status code, as indicated by the first digit, and treat any unknown response as being equivalent to the x00 status code of that class. For example, if an unknown status code of 421 is received by the client, it can safely assume that there was something wrong with its request and treat the response as if it had received a 400 status code. In such cases, user agents are encouraged to present the entity returned with the response to the user, since that entity is likely to include human-readable information which will explain the unusual status.

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

6.2.1 Informational 1xx

This class of status codes indicates a provisional response, consisting only of the Status-Line and optional headers, and is terminated by an empty line. HTTP/1.0 does not define any 1xx status codes and they are not a valid response to a standard HTTP/1.0 request. However, they may be useful for experimental applications which are outside the scope of this specification.

6.2.2 Successful 2xx

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

200 OK

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

GET an entity corresponding to the requested resource is being sent in the response;

HEAD the response must only contain the header information and no Entity-Body;

POST an entity describing or containing the result of the action;

PUT, DELETE, LINK, UNLINK
an entity describing the result of the action;

If the entity corresponds to a resource, the response may include a Location header field giving the actual location of that specific resource for later reference.

201 Created

The request has been fulfilled and resulted in a new resource being created. The newly created resource can be referenced by the URI(s) returned in the URI-header field of the response, with the most specific URL for the resource given by a Location header field. The origin server is encouraged, but not obliged, to actually create the resource before using this Status-Code. If the action cannot be carried out immediately, or within a clearly defined timeframe, the server should respond with 202 (accepted) instead.

Of the methods defined by this specification, only PUT and POST can create a resource.

202 Accepted

The request has been accepted for processing, but the processing has not been completed. The request may or may not eventually be acted upon, as it may be disallowed when processing actually takes place. There is no facility for re-sending a status code from an asynchronous operation such as this.

The 202 response is intentionally non-committal. Its purpose is to allow a server to accept a request for some other process (perhaps a batch-oriented process that is only run once per day) without requiring that the user agent's connection to the server persist until the process is completed. The entity returned with this response should include an indication of the request's current status and either a pointer to a status monitor or some estimate of when the user can expect the request to be fulfilled.

203 Non-Authoritative Information

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

204 No Content

The server has fulfilled the request but there is no new information to send back. If the client is a user agent, it should not change its document view from that which caused the request to be generated. This response is primarily intended to allow input for scripts or other actions to take place without causing a change to the user agent's active document view. The response may include new metainformation in the form of entity headers, which should apply to the document currently in the user agent's active view.

6.2.3 Redirection 3xx

This class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request. The action required can sometimes be carried out by the user agent without interaction with the user, but it is strongly recommended that this only take place if the method used in the request is idempotent (GET or HEAD). A user agent should never automatically redirect a request more than 5 times, since such redirections usually indicate an infinite loop.

300 Multiple Choices

The requested resource is available at one or more locations and a preferred location could not be determined via content negotiation. Unless it was a HEAD request, the response should include an entity containing a list of resource characteristics and locations from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type given in the Content-Type header field. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice may be performed automatically. If the server has a preferred choice, it should include its URL in a Location field; user agents not capable of complex selection may use the Location value for automatic redirection.

301 Moved Permanently

The requested resource has been assigned a new permanent URI and any future references to this resource should be done using one of the returned URIs. Clients with link editing capabilities are encouraged to automatically relink references to the Request-URI to one or more of the new references returned by the server, where possible.

If the new URI is a single location, its URL must be given by the Location field in the response. If more than one URI exists for the resource, the primary URL should be given in the Location field and the other URIs given in one or more URI-header fields. The Entity-Body of the response should contain a short hypertext note with a hyperlink to the new URI(s).

If the 301 status code is received in response to a request using the PUT, POST, or DELETE methods, the user agent must not automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

302 Moved Temporarily

The requested resource resides temporarily under a different URI. Since the redirection may be altered on occasion, the client should continue to use the Request-URI for future requests.

If the new URI is a single location, its URL must be given by the Location field in the response. If more than one URI exists for the resource, the primary URL should be given in the Location field and the other URIs given in one or more URI-header fields. The Entity-Body of the response should contain a short hypertext note with a hyperlink to the new URI(s).

If the 302 status code is received in response to a request using the PUT, POST, or DELETE methods, the user agent must not automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

303 See Other

The requested resource resides under a different URI and should be accessed using a GET method on that resource. This method exists primarily to allow the output of a POST-activated script to redirect the user agent to a selected resource. The new resource is not a replacement reference for the original Request-URI.

If the new URI is a single location, its URL must be given by the Location field in the response. If more than one URI exists for the resource, the primary URL should be given in the Location field and the other URIs given in one or more URI-header fields. The Entity-Body of the response should contain a short hypertext note with a hyperlink to the new URI(s).

304 Not Modified

If the client has performed a conditional GET request and access is allowed, but the document has not been modified since the date and time specified in the If-Modified-Since field, the server shall respond with this status code and not send an Entity-Body to the client. Header fields contained in the response should only include information which is relevant to cache managers and which may have changed independently of the entity's Last-Modified date. Examples of relevant header fields include: Date, Server, and Expires.

6.2.4 Client Error 4xx

The 4xx class of status codes is intended for cases in which the client seems to have erred. If the client has not completed the request when a 4xx code is received, it should immediately cease sending data to the server. Except when responding to a HEAD request, the server is encouraged to include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. These status codes are applicable to any request method.

> ***Note:** If the client is sending data, server implementations on TCP should be careful to ensure that the client acknowledges receipt of the packet(s) containing the response prior to closing the input connection. If the client continues sending data to the server after the close, the server's controller will send a reset packet to the client, which may erase the client's unacknowledged input buffers before they can be read and interpreted by the HTTP application.*

400 Bad Request

The request could not be understood by the server due to it having a malformed syntax. The client is discouraged from repeating the request without modifications.

401 Unauthorized

The request requires user authentication. The response must include a WWW-Authenticate header field (Section 8.30) containing a challenge applicable to the requested resource. The client may repeat the request with a suitable Authorization header field. HTTP access authentication is explained in Section 10.

402 Payment Required

This code is not currently supported, but is reserved for future use.

403 Forbidden

The server understood the request, but is refusing to perform the request because of an unspecified reason. Authorization will not help and the request should not be repeated. This status code can be used if the server does not want to make public why the request has not been fulfilled.

404 Not Found

The server has not found anything matching the Request-URI. No indication is given of whether the condition is temporary or permanent. If the server does not wish to make this information available to the client, the status code 403 (forbidden) can be used instead. The 410 (gone) status code should be used if the server knows, through some internally configurable mechanism, that an old resource is permanently unavailable and has no forwarding address.

405 Method Not Allowed

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

406 None Acceptable

The server has found a resource matching the Request-URI, but not one that satisfies the conditions identified by the Accept and Accept-Encoding request headers. Unless it was a HEAD request, the response should include an entity containing a list of resource characteristics and locations from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type given in the Content-Type header field. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice may be performed automatically.

407 Proxy Authentication Required

This code is reserved for future use. It is similar to 401 (unauthorized), but indicates that the client must first authenticate itself with the proxy. HTTP/1.0 does not provide a means for proxy authentication.

408 Request Timeout

The client did not produce a request within the time that the server was prepared to wait. The client may repeat the request without modifications at any later time.

409 Conflict

The request could not be completed due to a conflict with the current state of the resource. This code is only allowed in situations where it is expected that the user may be able to resolve the conflict and resubmit the request. The response body should include enough information for the user to recognize the source of the conflict. Ideally, the response entity would include enough information for the user or user-agent to fix the problem; however, that may not be possible and is not required.

Conflicts are most likely to occur in response to a PUT request. If versioning is being used and the entity being PUT includes changes to a resource which conflict with those made by an earlier (third-party) request, the server may use the 409 response to indicate that it can't complete the PUT. In this case, the response entity may contain a list of the differences between the two versions.

410 Gone

The requested resource is no longer available at the server and no forwarding address is known. This condition should be considered permanent. Clients with link editing capabilities are encouraged to delete references to the Request-URI after user approval. If the server does not know, or has no facility to determine, whether or not the condition is permanent, the status code 404 (not found) should be used instead.

The 410 response is primarily intended to assist the task of web maintenance by notifying the recipient that the resource is intentionally unavailable and that the server owners desire that remote links to that resource be removed. Such an event is common for limited-time, promotional services and for resources belonging to individuals no longer working at the server's site. It is not necessary to mark all permanently unavailable resources as “gone” or to keep the mark for any length of time — that is left to the discretion of the server owner.

411 Authorization Refused

The request credentials provided by the client were rejected by the server or insufficient to grant authorization to access the resource. This is similar to the 403 (forbidden) response, but allows more information to be provided to the user. The content of the response should contain a description of the problem and may suggest corrective action. HTTP access authentication is explained in Section 10.

The response must include a WWW-Authenticate header field (Section 8.30) containing a challenge applicable to the requested resource. If the challenge is different from that assumed by the last request, the client may repeat the request with a suitable Authorization header field after obtaining the user's approval.

6.2.5 Server Errors 5xx

Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has erred or is incapable of performing the request. If the client has not completed the request when a 5xx code is received, it should immediately cease sending data to the server. Except when responding to a HEAD request, the server is encouraged to include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. These response codes are applicable to any request method and there are no required header fields.

500 Internal Server Error

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

501 Not Implemented

The server does not support the functionality required to fulfill the request. This is the appropriate response when the server does not recognize the request method and is not capable of supporting it for any resource.

502 Bad Gateway

The server received an invalid response from the gateway or upstream server it accessed in attempting to fulfill the request.

503 Service Unavailable

The server is currently unable to handle the request due to a temporary overloading or maintenance of the server. The implication is that this is a temporary condition which will be alleviated after some delay. If known, the length of the delay may be indicated in a Retry-After header. If no Retry-After is given, the client should handle the response as it would for a 500 response.

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

504 Gateway Timeout

The server did not receive a timely response from the gateway or upstream server it accessed in attempting to complete the request.

6.3 Response Header Fields

The response header fields allow the server to pass additional information about the response which cannot be placed in the Status-Line. These header fields are not intended to give information about an Entity-Body returned in the response, but about the server itself.

```
Response-Header= Location               ; Section 8.19
               | Public                 ; Section 8.23
               | Retry-After            ; Section 8.25
               | Server                 ; Section 8.26
               | WWW-Authenticate       ; Section 8.30
```

Response-Header field names can be extended only via a change in the protocol version. Unknown header fields are treated as Entity-Header fields.

7. Entity

Full-Request and Full-Response messages may transfer an entity within some requests and responses. An entity consists of Entity-Header fields and (usually) an Entity-Body. In this section, both *sender* and *recipient* refer to either the client or the server, depending on who sends and who receives the entity.

7.1 Entity Header Fields

Entity-Header fields define optional metainformation about the Entity-Body or, if no body is present, about the resource identified by the request.

```
Entity-Header    = Allow                      ; Section 8.5
                 | Content-Encoding           ; Section 8.7
                 | Content-Language           ; Section 8.8
                 | Content-Length             ; Section 8.9
                 | Content-Transfer-Encoding  ; Section 8.10
                 | Content-Type               ; Section 8.11
                 | Expires                    ; Section 8.13
                 | Last-Modified              ; Section 8.17
                 | Link                       ; Section 8.18
                 | Title                      ; Section 8.27
                 | URI-header                 ; Section 8.28
                 | extension-header

extension-header = HTTP-header
```

The extension-header mechanism allows additional Entity-Header to be defined without changing the protocol, but these fields cannot be assumed to be recognizable by the recipient. Unknown header fields should be ignored by the recipient and forwarded by proxies.

7.2 Entity Body

The entity-body (if any) sent with an HTTP/1.0 request or response is in a format and encoding defined by the Entity-Header fields.

```
Entity-Body      = *OCTET
```

An entity-body is included with a request message only when the request method calls for one. This specification defines two request methods, "POST" and "PUT", that allow an entity-body. In general, the presence of an entity-body in a request is signaled by the inclusion of a Content-Length and/or Content-Transfer-Encoding header field in the request message headers. HTTP/1.0 requests containing content must include a valid Content-Length header field.

For response messages, whether or not an entity-body is included with a message is dependent on both the request method and the response code. All responses to the HEAD request method must not include a body, even though the presence of content header fields may lead one to believe they do. The responses 204 (no content) and 304 (not modified) must not include a message body.

7.2.1 Type

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

```
entity-body <-
   Content-Transfer-Encoding( Content-Encoding( Content-Type ) )
```

The default for both encodings is none (i.e., the identity function). A Content-Type specifies the media type of the underlying data. A Content-Encoding may be used to indicate any additional encoding mechanisms applied to the type, usually for the purpose of data compression, that is a property of the resource requested. A Content-Transfer-Encoding may be applied by a transport agent to ensure safe and proper transfer of the message. Note that the Content-Transfer-Encoding is a property of the message, not of the resource.

The Content-Type header field has no default value. If and **only if** the media type is not given by a Content-Type header, as is always the case for Simple-Response messages, the receiver may attempt to guess the media type via inspection of its content and/or the name extension(s) of the URL used to specify the resource. If the media type remains unknown, the receiver should treat it as type "`application/octet-stream`".

7.2.2 Length

When an Entity-Body is included with a message, the length of that body may be determined in one of several ways. If a Content-Length header field is present, its value in bytes represents the length of the Entity-Body. Otherwise, the body length is determined by the Content-Type (for types with an explicit end-of-body delimiter), the Content-Transfer-Encoding (for packetized encodings), or the closing of the connection by the server.

Closing the connection cannot be used to indicate the end of a request body, since it leaves no possibility for the server to send back a response. Furthermore, there is no guarantee that an HTTP/1.0 server will recognize types with an explicit end-of-body delimiter, and there is no packetized Content-Transfer-Encoding defined for HTTP/1.0. Therefore, HTTP/1.0 requests containing content must include a valid Content-Length header field. If a request contains an entity body and Content-Length is not specified, and the server does not recognize or cannot calculate the length from other fields, then the server should send a 400 (bad request) response.

> ***Note:*** *Some older servers supply an invalid Content-Length when sending a document that contains server-side includes dynamically inserted into the data stream. It must be emphasized that this will not be tolerated by future versions of HTTP. Unless the client knows that it is receiving a response from a compliant server, it should not depend on the Content-Length value being correct.*

8. Header Field Definitions

This section defines the syntax and semantics of all standard HTTP/1.0 header fields. 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.

8.1 Accept

The Accept header field can be used to indicate a list of media ranges which are acceptable as a response to the request. 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 set of ranges given by the client should represent what types are acceptable given the context of the request. The Accept field should only be used when the request is specifically limited to a set of desired types, as in the case of a request for an in-line image, or to indicate qualitative preferences for specific media types.

The field may be folded onto several lines and more than one occurrence of the field is allowed, with the semantics being the same as if all the entries had been in one field value.

```
Accept          = "Accept" ":" #(
                       media-range
                       [ ";" "q" "=" qvalue ]
                       [ ";" "mxb" "=" 1*DIGIT ] )
media-range     = ( "*/*"
                |    ( type "/" "*" )
                |    ( type "/" subtype )
                  ) *( ";" parameter )
```

The parameter q is used to indicate the quality factor, which represents the user's preference for that range of media types. The parameter mxb gives the maximum acceptable size of the Entity-Body, in decimal number of octets, for that range of media types. Section 9 describes the content negotiation algorithm which makes use of these values. The default values are: q=1 and mxb=undefined (i.e., infinity).

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 is present, then it is assumed that the client accepts *all* media types with quality factor 1. This is equivalent to the client sending the following accept header field:

```
Accept: */*; q=1
```

or

```
Accept: */*
```

A more elaborate example is

```
Accept: text/plain; q=0.5, text/html,
        text/x-dvi; q=0.8; mxb=100000, 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 Entity-Body in text/x-dvi if the entity is less than 100000 bytes, otherwise send text/plain.”

> ***Note:*** *In earlier versions of this document, the* mxs *parameter defined the maximum acceptable delay in seconds before the response would arrive. This has been removed as the server has no means of obtaining a useful reference value. However, this does not prevent the client from internally measuring the response time and optimizing the* Accept *header field accordingly.*

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;version=2.0, */*
```

have the following precedence:

```
1) text/html;version=2.0
2) text/html
3) text/*
4) */*
```

The quality value 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;version=2.0,
        */*;q=0.5
```

would cause the following values to be associated:

```
text/html;version=2.0          = 1
text/html                      = 0.7
text/plain                     = 0.3
image/jpeg                     = 0.5
text/html;level=3              = 0.7
```

It must be emphasized that the Accept field should only be used when it is necessary to restrict the response media types to a subset of those possible or when the user has been permitted to specify qualitative values for ranges of media types. If no quality factors have been set by the user, and the context of the request is such that the user agent is capable of saving the entity to a file if the received media type is unknown, then the only appropriate value for Accept is "*/*".

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

8.2 Accept-Charset

The Accept-Charset request header field can be used to indicate a list of preferred character set encodings other than the default US-ASCII and ISO-8859-1. This field allows clients capable of understanding more comprehensive or special-purpose character set encodings to signal that capability to a server which is capable of representing documents in those character set encodings.

```
Accept-Charset    = "Accept-Charset" ":" #charset
```

Character set encoding values are described in Section 3.5. An example is

```
Accept-Charset: iso-8859-5, unicode-1-1
```

The value of this field should not include "US-ASCII" or "ISO-8859-1", since those values are always assumed by default. If a resource is only available in a character set encoding other than the defaults, and that character set encoding is not listed in the Accept-Charset field, it is only acceptable for the server to send the entity if the character set encoding can be identified by an appropriate charset parameter on the media type or within the format of the media type itself.

> ***Note:*** *User agents are not required to be able to render the characters associated with the ISO-8859-1 character set encoding. However, they must be able to interpret their meaning to whatever extent is required to properly handle messages in that character set encoding.*

8.3 Accept-Encoding

The Accept-Encoding request header field is similar to Accept, but restricts the encoding-mechanism values which are acceptable in the response.

```
Accept-Encoding          = "Accept-Encoding" ":"
                           #( encoding-mechanism )
```

An example of its use is

```
Accept-Encoding: compress, gzip
```

If no Accept-Encoding field is present in a request, the server should assume that the client will accept any encoding-mechanism.

8.4 Accept-Language

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.

```
Accept-Language          = "Accept-Language" ":"
                           #( language-tag [ ";" "ql" "=" qvalue ] )
```

The language-tag is described in Section 3.8. Each language may be given an associated quality value which represents an estimate of the user's comprehension of that language. The quality value defaults to "ql=1" (100% comprehension) for listed languages. This value may be used in the server's content negotiation algorithm (Section 9). For example,

```
Accept-Language: da, en-gb;ql=0.8, de;ql=0.55
```

would mean: "I prefer Danish, but will accept British English (with 80% comprehension) or German (with a 55% comprehension)."

If the server cannot fulfill the request with one or more of the languages given, or if the languages only represent a subset of a multi-linguistic Entity-Body, it is acceptable to serve the request in an unspecified language. This is equivalent to asssigning a quality value of "ql=0.001" to any unlisted language.

If no Accept-Language header is present in the request, the server should assume that all languages are equally acceptable.

> ***Note:*** *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.*

8.5 Allow

The Allow header field lists the set of methods supported by the resource identified by the Request-URI. The purpose of this field is strictly to inform the recipient of valid methods associated with the resource. An Allow header field must be present in a 405 (method not allowed) response. The Allow header field is not permitted in a request using the POST method, and thus should be ignored if it is received as part of a POST entity.

```
Allow          = "Allow" ":" #method
```

Example of use:

```
Allow: GET, HEAD, PUT
```

This field cannot prevent a client from trying other methods. However, the indications given by the Allow field value should be followed. This field has no default value; if left undefined, the set of allowed methods is defined by the origin server at the time of each request.

The Allow header field may be provided with a PUT request to recommend the methods to be supported by the new or modified resource. The server is not required to support these methods and should include an Allow header in the response giving the actual supported methods.

A proxy must not modify the allow header even if it does not understand all the methods specified, since the user agent may have other means of communicating with the origin server.

The Allow header field does not indicate what methods are implemented at the server level. Servers must use the Public response header field (Section 8.23) if they wish to describe what methods are implemented on the server as a whole.

8.6 Authorization

A user agent that wishes to authenticate itself with a server—usually, but not necessarily, after receiving a 401 or 411 response—may do so by including an Authorization header field with the request. The Authorization field value consists of credentials containing the authentication information of the user agent for the realm of the resource being requested.

```
Authorization  = "Authorization" ":" 1#credentials
```

HTTP access authentication is described in Section 10. If a request is authenticated and a realm specified, the same credentials should be valid for all other requests within this realm, until the server indicates otherwise with a 411 (authorization refused) response.

8.7 Content-Encoding

The Content-Encoding header field is used as a modifier to the media-type. When present, its value indicates what additional encoding mechanisms have been applied to the resource, and thus what decoding mechanisms must be applied in order to obtain the media-type referenced by the Content-Type header field. The 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#encoding-mechanism
```

Encoding mechanisms are defined in Section 3.6. An example of its use is

```
Content-Encoding: gzip
```

The Content-Encoding is a characteristic of the resource identified by the Request-URI. Typically, the resource is stored with this encoding and is only decoded before rendering or analogous usage.

If multiple encodings have been applied to a resource, the encoding-mechanisms 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.

8.8 Content-Language

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

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

Language tags are defined in Section 3.8. The primary purpose of Content-Language is to allow a selective consumer to identify and differentiate resources according to the consumer's own preferred language. Thus, if the body content is intended only for a Danish-literate audience, the appropriate field is

```
Content-Language: dk
```

If no Content-Language is specified, the default is that the content is intended for all language audiences. This may 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 should only include "en".

Content-Language may be applied to any media type — it should not be limited to textual documents.

8.9 Content-Length

The Content-Length header field indicates the size of the Entity-Body, in decimal number of octets, sent to the recipient or, in the case of the HEAD method, the size of the Entity-Body that would have been sent had the request been a GET.

```
Content-Length    = "Content-Length" ":" 1*DIGIT
```

An example is

```
Content-Length: 3495
```

Although it is not required, applications are strongly encouraged to use this field to indicate the size of the Entity-Body to be transferred, regardless of the media type of the entity.

Any Content-Length greater than or equal to zero is a valid value. Section 7.2.2 describes how to determine the length of an Entity-Body if a Content-Length is not given.

> ***Note:*** *The meaning of this field is significantly different from the corresponding definition in MIME, where it is an optional field used within the "message/external-body" content-type. In HTTP, it should be used whenever the entity's length can be determined prior to being transferred.*

8.10 Content-Transfer-Encoding

The Content-Transfer-Encoding (CTE) header indicates what (if any) type of transformation has been applied to the entity in order to safely transfer it between the sender and the recipient. This differs from the Content-Encoding in that the CTE is a property of the message, not of the original resource.

```
Content-Transfer-Encoding  = "Content-Transfer-Encoding" ":" transfer-encoding
```

Transfer encodings are defined in Section 3.7. Because all HTTP transactions take place on an 8-bit clean connection, the default Content-Transfer-Encoding for all messages is binary. However, HTTP may be used to transfer MIME messages which already have a defined CTE. An example is:

```
Content-Transfer-Encoding: quoted-printable
```

Many older HTTP/1.0 applications do not understand the Content-Transfer-Encoding header. However, since it may appear in any MIME message (i.e., entities retrieved via a gateway to a MIME-conformant protocol), future HTTP/1.0 applications must understand it upon receipt. Gateways are the only HTTP applications that would generate a CTE.

8.11 Content-Type

The Content-Type 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 3.4. An example of the field is

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

The Content-Type header field has no default value. Further discussion of methods for identifying the media type of an entity is provided in Section 7.2.1.

8.12 Date

The Date header represents the date and time at which the message was originated, having the same semantics as orig-date in RFC 822.The field value is an HTTP-date, as described in Section 3.3.

```
Date           = "Date" ":" HTTP-date
```

An example is

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

If a message is received via direct connection with the user agent (in the case of requests) or the origin server (in the case of responses), then the default date can be assumed to be the current date at the receiving end. However, since the date—as it is believed by the origin—is important for evaluating cached responses, origin servers should always include a Date header. Clients should only send a Date header field in messages that include an entity body, as in the case of the PUT and POST requests, and even then it is optional. A received message which does not have a Date header field should be assigned one by the receiver if and only if the message will be cached by that receiver or gatewayed via a protocol which requires a Date.

Only one Date header field is allowed per message. In theory, the date should represent the moment just before the entity is generated. In practice, the date can be generated at any time during the message origination without affecting its semantic value.

> ***Note:*** *An earlier version of this document incorrectly specified that this field should contain the creation date of the enclosed* Entity-Body. *This has been changed to reflect actual (and proper) usage.*

8.13 Expires

The Expires field gives the date/time after which the entity should be considered stale. This allows information providers to suggest the volatility of the resource. Caching clients, including proxies, must not cache this copy of the resource beyond the date given, unless its status has been updated by a later check of the origin server. The presence of an Expires field does not imply that the original resource will change or cease to exist at, before, or after that time. However, information providers that know or even suspect that a resource will change by a certain date are strongly encouraged to include an Expires header with that date. The format is an absolute date and time as defined by HTTP-date in Section 3.3.

```
Expires        = "Expires" ":" HTTP-date
```

An example of its use is

```
Expires: Thu, 01 Dec 1994 16:00:00 GMT
```

The Expires field has no default value. If the date given is equal to or earlier than the value of the Date header, the recipient must not cache the enclosed entity. If a resource is dynamic by nature, as is the case with many data-producing processes, copies of that resource should be given an appropriate Expires value which reflects that dynamism.

The Expires field cannot be used to force a user agent to refresh its display or reload a resource; its semantics apply only to caching mechanisms, and such mechanisms need only check a resource's expiration status when a new request for that resource is initiated.

User agents often have history mechanisms, such as "Back" buttons and history lists, which can be used to redisplay an entity retrieved earlier in a session. The Expires field does not apply to history mechanisms. If the entity is still in storage, a history mechanism should display it even if the entity has expired.

> ***Note:*** *Applications are encouraged to be tolerant of bad or misinformed implementations of the Expires header. In particular, recipients may wish to recognize a delta-seconds value (any decimal integer) as representing the number of seconds after receipt of the message that its contents should be considered expired. Likewise, a value of zero (0) or an invalid date format should be considered equivalent to an "expires immediately." Although these values are not legitimate for HTTP/1.0, a robust implementation is always desirable.*

8.14 Forwarded

The Forwarded header is to be used by proxies to indicate the intermediate steps between the user agent and the server on requests, and between the origin server and the client on responses. It is analogous to the "Received" field of RFC 822 [8] and is intended to be used for tracing transport problems and avoiding request loops.

```
Forwarded      = "Forwarded" ":" #( "by" URI [ "(" product ")" ]
                  [ "for" FQDN ] )

FQDN           = <Fully-Qualified Domain Name>
```

For example, a message could be sent from a client on *ptsun00.cern.ch* to a server at *www.ics.uci.edu* port 80, via an intermediate HTTP proxy at *info.cern.ch* port 8000. The request received by the server at *www.ics.uci.edu* would then have the following Forwarded header field:

```
Forwarded: by http://info.cern.ch:8000/ for ptsun00.cern.ch
```

Multiple Forwarded header fields are allowed and should represent each proxy that has forwarded the message. It is strongly recommended that proxies used as a portal through a network firewall do *not*, by default, send out information about the internal hosts within the firewall region. This information should only be propagated if explicitly enabled. If not enabled, the for token and FQDN should not be included in the field value, and any Forwarded headers already present in the message (those added behind the firewall) should be removed.

8.15 From

The From header field, if given, should contain an Internet e-mail address for the human user who controls the requesting user agent. The address should be machine-usable, as defined by mailbox in RFC 822 [8] (as updated by RFC 1123 [7]):

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

An example is:

```
From: webmaster@w3.org
```

This header field may be used for logging purposes and as a means for identifying the source of invalid or unwanted requests. It should not be used as an insecure form of access protection. The interpretation of this field is that the request is being performed on behalf of the person given, who accepts responsibility for the method performed. In particular, robot agents should include this header so that the person responsible for running the robot can be contacted if problems occur on the receiving end.

The Internet e-mail address in this field does not have to correspond to the Internet host which issued the request. For example, when a request is passed through a proxy the original issuer's address should be used.

> ***Note:*** *The client should not send the From header field without the user's approval, as it may conflict with the user's privacy interests or their site's security policy. It is strongly recommended that the user be able to disable, enable, and modify the value of this field at any time prior to a request.*

8.16 If-Modified-Since

The If-Modified-Since header field is used with the GET method to make it conditional: if the requested resource has not been modified since the time specified in this field, a copy of the resource will not be returned from the server; instead, a "304 Not Modified" response will be returned without any Entity-Body.

```
If-Modified-Since  = "If-Modified-Since" ":" HTTP-date
```

An example of the field is:

```
If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

A conditional GET method requests that the identified resource be transferred only if it has been modified since the date given by the If-Modified-Since header. The algorithm for determining this includes the following cases:

a) If the request would normally result in anything other than a "200 OK" status, or if the passed If-Modified-Since date is invalid, the response is exactly the same as for a normal GET.

b) If the resource has been modified since the If-Modified-Since date, the response is exactly the same as for a normal GET.

c) If the resource has not been modified since the If-Modified-Since date, the server shall return a "304 Not Modified" response.

The purpose of this feature is to allow efficient updates of cached information with a minimum amount of transaction overhead.

> ***Note:*** *The same functionality can be obtained, though with much greater overhead, by issuing a HEAD request and following it with a GET request if the server indicates that the entity has been modified.*

8.17 Last-Modified

The Last-Modified header field indicates the date and time at which the sender believes the resource was last modified. The exact semantics of this field are defined in terms of how the receiver should interpret it: if the receiver has a copy of this resource which is older than the date given by the Last-Modified field, that copy should be considered stale.

```
Last-Modified      = "Last-Modified" ":" HTTP-date
```

An example of its use is

```
Last-Modified: Tue, 15 Nov 1994 12:45:26 GMT
```

The exact meaning of this header field depends on the implementation of the sender and the nature of the original resource. For files, it may be just the file system last-mod date. For entities with dynamically included parts, it may be the most recent of the set of last-modify times for its component parts. For database gateways, it may be the last-update timestamp of the record. For virtual objects, it may be the last time the internal state changed.

8.18 Link

The Link header provides a means for describing a relationship between the entity and some other resource. An entity may include multiple Link values. Links at the metainformation level typically indicate relationships like hierarchical structure and navigation paths. The Link field is semantically equivalent to the <LINK> element in HTML [4].

```
Link            = "Link" ":" #("<" URI ">"
                  [ ";" "rel" "=" relationship ]
                  [ ";" "rev" "=" relationship ]
                  [ ";" "title" "=" quoted-string ] )

relationship    = sgml-name
                | ( <"> sgml-name *( SP sgml-name) <"> )

sgml-name       = ALPHA *( ALPHA | DIGIT | "." | "-" )
```

Relation values are not case-sensitive and may be extended within the constraints of the sgml-name syntax. There are no predefined link relationship values for HTTP/1.0. The title parameter may be used to label the destination of a link such that it can be used as identification within a human-readable menu. Examples of usage include:

```
Link: <http://www.cern.ch/TheBook/chapter2>; rel="Previous"

Link: <mailto:timbl@w3.org>; rev="Made"; title="Tim Berners-Lee"
```

The first example indicates that the entity is previous to chapter2 in a logical navigation path. The second indicates that the person responsible for making the resource available is identified by the given e-mail address.

8.19 Location

The Location response header field defines the exact location of the resource that was identified by the Request-URI. For 2xx responses, the location should be the URL needed to retrieve that same resource again (i.e., if variants of that resource are available, the value of the Location field should locate the variant chosen by the server if it has its own specific URL). For 3xx responses, the location should indicate the server's preferred URL for automatic redirection to the resource. Only one absolute URL is allowed.

```
Location        = "Location" ":" absoluteURI
```

An example is

```
Location: http://www.w3.org/hypertext/WWW/NewLocation.html
```

If no base URL is provided by or within the entity, the value of the Location field should be used as the base for resolving relative URLs [10].

8.20 MIME-Version

HTTP is not a MIME-conformant protocol (see Appendix C). However, HTTP/1.0 messages may include a single MIME-Version header field to indicate what version of the MIME protocol was used to construct the message. Use of the MIME-Version header field should indicate that the message is in full compliance with the MIME protocol (as defined in [6]). Unfortunately, current versions of HTTP/1.0 clients and servers use this field indiscriminately, and thus receivers must not take it for granted that the message is indeed in full compliance with MIME. Gateways are responsible for ensuring this compliance (where possible) when exporting HTTP messages to strict MIME environments. Future HTTP/1.0 applications must only use MIME-Version when the message is intended to be MIME-conformant.

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

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

8.21 Orig-URI

The Orig-URI request header field allows the client to specify, for the server’s benefit, the original Uniform Resource Identifier (Section 3.2) of the resource being requested, as it was obtained from the user or the referring resource. This allows a server to differentiate between internally-ambiguous URLs (such as the root "/" URL of a server harboring multiple virtual hostnames), to learn about new URNs used to reference resources on the server, and to provide some additional assistance in identifying and redirecting moved resources and resource fragments.

```
Orig-URI          = "Orig-URI" ":" absoluteURI [ "#" fragment ]
```

Example:

```
Orig-URI: http://www.w3.org/
```

The URI must be in absolute form and should include the fragment if one is given to the client. It should include exactly what was referenced by the Referer resource, with the exception that a relative reference must first be resolved to its absolute form.

8.22 Pragma

The Pragma message header field is used to specify directives that should be applied to all intermediaries along the request/response chain. The directives typically specify behavior intended to prevent intermediate proxies or caches from adversely interfering with the request or response. All pragma directives specify optional behavior from the viewpoint of the protocol; however, some systems may require that behavior be consistent with the directives. HTTP/1.0 defines semantics for the “`no-cache`” and “`max-age`” directives.

```
Pragma            = "Pragma" ":" #pragma-directive

pragma-directive  = "no-cache"
                  | "max-age" "=" delta-seconds
                  | extension-pragma
extension-pragma  = token [ "=" word ]
```

When the "no-cache" directive is present in a request message, a caching intermediary should forward the request toward the origin server even if it has a cached copy of what is being requested. This allows a client to insist upon receiving an authoritative response to its request. It also allows a client to refresh a cached copy which is known to be corrupted or stale.

When the "no-cache" directive is present in a response message, caching intermediaries are requested to not cache this response. This allows an origin server to state that the message is intended for only one recipient and may not be a valid response for other requests.

When the "max-age" directive is present in a request message, a caching intermediary should forward the request toward the origin server if it has no cached copy, or refresh its cached copy if it is older than the age value given (in seconds) prior to returning a response. A cached copy's "age" is determined by the cached message's Date header field, or the equivalent as stored by the cache manager. In most cases, a cached copy can be refreshed by forwarding a conditional GET request toward the origin server with the stored message's Date value in the If-Modified-Since field. If a 304 (not modified) response is received, the cache should replace the cached message's Date with that of the 304 response and send this refreshed message as the response. Any other response should be forwarded directly to the requestor and, depending on the response code and the discretion of the cache manager, may replace the message in the cache.

When the "max-age" directive is present in a cached response message, a caching intermediary should refresh the message if it is older than the age value given (in seconds) at the time of a new request for that resource. The behavior should be equivalent to what would occur if the request had included that pragma directive. If both the new request and the cached message have max-age specified, then the lesser of the two values should be used.

Pragma directives must be passed through by a proxy, regardless of their significance to that proxy, since the directives may be applicable to all intermediaries along the request/response chain. It is not possible to specify a pragma for a specific proxy; however, any pragma directive not relevant to a proxy should be ignored.

Pragma directives do not apply to the end-points of a request/response chain. For example, a user agent's internal (non-shared) cache and/or history mechanism should ignore all pragma directives in received messages. Similarly, pragma directives are not applicable to the origin of a resource, though they may be applicable to a server's internal response cache.

8.23 Public

The Public response header field lists the set of non-standard methods supported by the server. The purpose of this field is strictly to inform the recipient of the capabilities of the server regarding unusual methods. The methods listed may or may not be applicable to the Request-URI; the Allow header field (Section 8.5) should be used to indicate methods allowed for a particular URI. This does not prevent a client from trying other methods. The field value should not include the methods predefined for HTTP/1.0 in Section 5.2.

```
Public         = "Public" ":" #method
```

Example of use:

```
Public: OPTIONS, MGET, MHEAD
```

This header field applies only to the server directly connected to the client (i.e., the nearest neighbor in a chain of connections). If the response passes through a proxy, the proxy must either remove the Public header field or replace it with one applicable to its own capabilities.

8.24 Referer

The Referer request header field allows the client to specify, for the server's benefit, the address (URI) of the resource from which the Request-URI was obtained. This allows a server to generate lists of back-links to resources for interest, logging, optimized caching, etc. It also allows obsolete or mistyped links to be traced for maintenance. The Referer field must not be sent if the Request-URI was obtained from a source that does not have its own URI, such as input from the user keyboard.

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

Example:

```
Referer: http://info.cern.ch/hypertext/DataSources/Overview.html
```

If a partial URI is given, it should be interpreted relative to the Request-URI. The URI must not include a fragment.

> ***Note:*** *Because the source of a link may be private information or may reveal an otherwise private information source, it is strongly recommended that the user be able to select whether or not the Referer field is sent. For example, a browser client could have a toggle switch for browsing openly/anonymously, which would respectively enable/disable the sending of Referer and From information.*

8.25 Retry-After

The Retry-After response header field can be used with a 503 (service unavailable) response to indicate how long the service is expected to be unavailable to the requesting client. The value of this field can be either an HTTP-date or an integer number of seconds (in decimal) after the time of the response.

```
Retry-After    = "Retry-After" ":" ( HTTP-date | delta-seconds )
```

Two examples of its use are

```
Retry-After: Wed, 14 Dec 1994 18:22:54 GMT

Retry-After: 120
```

In the latter example, the delay is 2 minutes.

8.26 Server

The Server response header field contains information about the software used by the origin server to handle the request. The field can contain multiple product tokens (Section 3.10) identifying the server and any significant subproducts. By convention, the product tokens are listed in order of their significance for identifying the application.

```
Server         = "Server" ":" 1*( product )
```

Example:

```
Server: CERN/3.0 libwww/2.17
```

If the response is being forwarded through a proxy, the proxy application must not add its data to the product list. Instead, it should include a Forwarded field (as described in Section 8.14).

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

8.27 Title

The Title header field indicates the title of the entity

```
Title          = "Title" ":" *text
```

An example of the field is

```
Title: Hypertext Transfer Protocol — HTTP/1.0
```

This field is isomorphic with the <TITLE> element in HTML [4].

8.28 URI

The URI-header field may contain some or all of the Uniform Resource Identifiers (Section 3.2) by which the Request-URI resource can be identified. There is no guarantee that the resource can be accessed using the URI(s) specified.

```
URI-header     = "URI" ":" #( "<" ( absoluteURI | relativeURI ) ">"
                   [ ";" vary ] *( ";" characteristic) )

vary           = "vary" "=" ( vary-dimension | ( <"> 1#vary-dimension <"> ) )
vary-dimension = "type" | "charset" | "language" | "encoding"
               |  "user-agent" | "version" | token

characteristic = ( "type={" media-type "}" )
               |  ( "language={" 1#language-tag "}" )
               |  ( "encoding={" 1#encoding-mechanism "}" )
               |  ( "length=" 1*DIGIT )
               |  ( "qs=" qvalue )
```

Any URI specified in this field can be either absolute or relative to the Request-URI.

If the Location header field is present in a 2xx response, its value defines an implicit URI header with the characteristic parameters defined by the associated Content-* header fields.

The URI-header may be used by a client performing a POST request to suggest a URI for the new entity. Whether or not the suggested URI is used is entirely up to the server to decide. In any case, the server's response must include the actual URI(s) of the new resource if one is successfully created (status 201).

If a URI refers to a set of variants, then the dimensions of that variance must be given with a vary parameter. One example is:

```
URI: <http://info.cern.ch/hypertext/WWW/TheProject.multi>;
     vary="type,language"
```

which indicates that the URI covers a group of entities that vary in media type and natural language. A request for that URI will result in a response that depends upon the client's request headers for Accept and Accept-Language. Similar dimensions exist for the Accept-Encoding, Accept-Charset, and User-Agent header fields, as demonstrated in the following example.

```
URI: <TheProject.ps>; vary="encoding,version";
     type={application/postscript},
     <TheProject.html>; vary="user-agent,charset,version";
     type={text/html},
     <TheProject.html3;v=25>; type={text/html; level=3}; qs=0.9
```

User agents may use this information to notify the user of additional formats.

The vary parameter has an important effect on cache management, particularly for caching intermediaries which service a diverse set of user agents. Since the response to one user agent may differ from the response to a second user agent if the two agents have differing request profiles, a caching intermediary must keep track of the content metainformation for resources with varying dimensions. Thus, the vary parameter tells the intermediary what entity headers must be part of the key for caching that URI. When the caching proxy gets a request for that URI, it must forward the request toward the origin server if the request profile includes a variant dimension that has not already been cached.

If the origin server provides the characteristics of each identified resource as part of the URI header, then the recipient may improve its cached response behavior by attempting to duplicate the content negotiation that would be provided by the server. This is not required by the protocol, but may improve the accuracy or timeliness of responses to the end-user.

8.29 User-Agent

The User-Agent field contains information about the user agent originating the request. This is for statistical purposes, the tracing of protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations. Although it is not required, user agents should always include this field with requests. The field can contain multiple product tokens (Section 3.10) identifying the agent and any subproducts which form a significant part of the user agent. By convention, the product tokens are listed in order of their significance for identifying the application.

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

Example:

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

The User-Agent field may include additional information within comments.

> ***Note:*** *Some current proxy applications append their product information to the list in the User-Agent field. This is no longer recommended, since it makes machine interpretation of these fields ambiguous. Instead, proxies should use the* Forwarded *header described in Section 8.14.*

8.30 WWW-Authenticate

The WWW-Authenticate header field must be included in 401 (unauthorized) and 411 (authorization refused) response messages. The field value consists of a challenge that indicates the authentication scheme and parameters applicable to the Request-URI.

```
WWW-Authenticate       = "WWW-Authenticate" ":" challenge
```

The HTTP access authentication process is described in Section 10.

9. Content Negotiation

Content negotiation is an optional feature of the HTTP protocol. It is designed to allow for selection of a preferred content representation, within a single request-response round-trip, and without intervention from the user. However, this may not always be desirable for the user and is sometimes unnecessary for the content provider. Implementors are encouraged to provide mechanisms whereby the amount of preemptive content negotiation, and the parameters of that negotiation, are configurable by the user and server maintainer.

The first step in the negotiation algorithm is for the server to determine whether or not there are any content variants for the requested resource. Content variants may be in the form of multiple preexisting entities or a set of dynamic conversion filters. These variants make up the set of entities which may be sent in response to a request for the given Request-URI. In most cases, there will only be one available form of the resource, and thus a single "variant".

For each variant form of the resource, the server identifies a set of quality values (Section 3.9) which act as weights for measuring the desirability of that resource as a response to the current request. The calculated weights are all real numbers in the range 0 through 1, where 0 is the minimum and 1 the maximum value. The maximum acceptable bytes for each media range and the size of the resource variant are also factors in the equation.

The following parameters are included in the calculation:

qs
: Source quality is measured by the content provider as representing the amount of degradation from the original source. For example, a picture originally in JPEG form would have a lower qs when translated to the XBM format, and much lower qs when translated to an ASCII-art representation. Note, however, that this is a function of the source — an original piece of ASCII-art may degrade in quality if it is captured in JPEG form. The qs value should be assigned to each variant by the content provider; if no qs value has been assigned, the default is generally "qs=1". A server may define its own default qs value based on the resource characteristics, but only if individual resources can override those defaults.

qe
: Encoding quality is measured by comparing the variant's applied encoding-mechanisms (Section 3.6) to those listed in the request message's Accept-Encoding field. If the variant has no assigned Content-Encoding, or if no Accept-Encoding field is present, the value assigned is "qe=1". If all of the variant's content encodings are listed in the Accept-Encoding field, then the value assigned is "qe=1". If any of the variant's content encodings are not listed in the provided Accept-Encoding field, then the value assigned is "qe=0.001".

qc
: Charset quality is measured by comparing the variant media-type's charset parameter value (if any) to those character set encodings (Section 3.5) listed in the request message's Accept-Charset field. If the variant's media-type has no charset parameter, or the variant's charset is US-ASCII or ISO-8859-1, or if no Accept-Charset field is present, then the value assigned is "qc=1". If the variant's charset is listed in the Accept-Charset field, then the value assigned is "qc=1". Otherwise, if the variant's charset is not listed in the provided Accept-Encoding field, then the value assigned is "qc=0.001".

ql
: Language quality is measured by comparing the variant's assigned language tag(s) (Section 3.8) to those listed in the request message's Accept-Language field. If no variant has an assigned Content-Language, or if no Accept-Language field is present, the value assigned is "ql=1". If at least one variant has an assigned content language, but the one currently under consideration does not, then it should be assigned the value "ql=0.5". If any of the variant's content languages are listed in the Accept-Language field, then the value assigned is the maximum of the "ql" parameter values for those language tags (Section 8.4); if there was no exact match and at least one of the Accept-Language field values is a complete subtag prefix of the content language tag(s), then the "ql" parameter value of the largest matching prefix is used. If none of the variant's content language tags or tag prefixes are listed in the provided Accept-Language field, then the value assigned is "ql=0.001".

q
: Media type quality is measured by comparing the variant's assigned media type (Section 3.4) to those listed in the request message's Accept field. If no Accept field is given, then the value assigned is "q=1". If at least one listed media range (Section 8.1) matches the variant's media type, then the "q" parameter value assigned to the most specific of those matched is used (e.g., "text/html;version=3.0" is more specific than "text/html", which is more specific than "text/*", which in turn is more specific than "*/*"). If no media range in the provided Accept field matches the variant's media type, then the value assigned is "q=0".

mxb
: The maximum number of bytes in an Entity-Body that the client will accept is also obtained from the matching of the variant's assigned media type to those listed in the request message's Accept field. If no Accept field is given, or if no media range in the provided Accept field matches the variant's media type, then the value assigned is "mxb=undefined" (i.e., infinity). Otherwise, the value used is that given to the "mxb" parameter in the media range chosen above for the q value.

bs
: The actual number of bytes in the Entity-Body for the variant when it is included in a response message. This should equal the value of Content-Length.

The mapping function is defined as:

```
Q(qs,qe,qc,ql, { if mxb=undefined,then (qs*qe*qc*ql*q)}
  q,mxb,bs)   ={ if mxb >= bs,    then (qs*qe*qc*ql*q)}
               { if mxb <  bs,    then 0             }
```

The variants with a maximal value for the Q function represent the preferred representation(s) of the entity; those with a Q values less than the maximal value are therefore excluded from further consideration. If multiple representations exist that only vary by Content-Encoding, then the smallest representation (lowest bs) is preferred.

If no variants remain with a value of Q greater than zero (0), the server should respond with a 406 (none acceptable) response message. If multiple variants remain with an equally high Q value, the server may either choose one from those available and respond with 200 (ok) or respond with 300 (multiple choices) and include an entity describing the choices. In the latter case, the entity should either be of type "text/html', such that the user can choose from among the choices by following an exact link, or of some type that would allow the user agent to perform the selection automatically.

The 300 (multiple choices) response can be given even if the server does not perform any winnowing of the representation choices via the content negotiation algorithm described above. Furthermore, it may include choices that were not considered as part of the negotiation algorithm and resources that may be located at other servers.

Servers that make use of content negotiated resources are strongly encouraged to include URI response headers which accurately describe the available variants and include the relevant parameters necessary for the client (user agent or proxy) to evaluate those variants.

The algorithm presented above assumes that the user agent has correctly implemented the protocol and is accurately communicating its intentions in the form of Accept-related header fields. The server may alter its response if it knows that the particular version of user agent software making the request has incorrectly or inadequately implemented these fields.

10. Access Authentication

HTTP provides a simple challenge-response authorization style which may be used by a server to challenge a client request and by a client to provide authentication information. It uses an extensible, case-insensitive token to identify the authentication scheme, followed by a semicolon-separated list of attribute-value pairs which carry the parameters necessary for achieving authentication via that scheme.

```
auth-scheme      = "basic" | token
auth-param       = token "=" quoted-string
```

The 401 (unauthorized) response message is used by an origin server to challenge the authorization of a user agent. This response must include a WWW-Authenticate header field containing a challenge applicable to the requested resource.

```
challenge        = auth-scheme 1*SP realm *( ";" auth-param )
realm            = "realm" "=" quoted-string
```

The realm attribute (case-insensitive) is required for all authentication schemes which issue a challenge. The realm value (case-sensitive), in combination with the root URL of the server being accessed, defines the protection space. These realms allow the protected resources on a server to be partitioned into a set of protection spaces, each with its own authentication scheme and/or authorization database. The realm value is a string, generally assigned by the origin server, which may have additional semantics specific to the authentication scheme.

A user agent that wishes to authenticate itself with a server—usually, but not necessarily, after receiving a 401 or 411 response—may do so by including an Authorization header field with the request. The Authorization field value consists of credentials containing the authentication information of the user agent for the realm of the resource being requested.

```
credentials      = auth-scheme [ 1*LWS encoded-cookie ] *(";" auth-param )
encoded-cookie   = <any valid base64 [6] encoded string,
                     except not limited to 76 char/line>
```

The domain over which credentials can be automatically applied by a user agent is determined by the authorization space. If a request is authenticated, the credentials can be reused for all other requests within that authorization space for a period of time determined by the authentication scheme, parameters, and/or user preference.

If the server does not wish to accept the credentials sent with a request, it should return either a 403 (forbidden) or 411 (authorization refused) response. In the latter case, the response must include a WWW-Authenticate header field containing the (possibly new) challenge applicable to the requested resource and an entity explaining the refusal.

The HTTP protocol does not restrict applications to this simple challenge-response mechanism for access authentication. Additional mechanisms may be used at the transport level, via message encapsulation, and/or with additional header fields specifying authentication information. However, these additional mechanisms are not defined by this specification.

Proxies must be completely transparent regarding user agent authentication. That is, they must forward the WWW-Authenticate and Authorization headers untouched. HTTP/1.0 does not provide a means for a client to be authenticated with a proxy.

> ***Note:*** *The names Proxy-Authenticate and Proxy-Authorization have been suggested as headers, analogous to WWW-Authenticate and Authorization, but applying only to the immediate connection with a proxy.*

10.1 Basic Authentication Scheme

The basic authentication scheme is based on the model that the client must authenticate itself with a user-ID and a password for each realm. The realm value should be considered an opaque string which can only be compared for equality with other realms. The server will service the request only if it can validate the user-ID and password for the domain of the Request-URI.

```
basic-challenge    = "Basic" SP realm
```

The client sends the user-ID and password, separated by a single colon ":" character, within a base64 [6] encoded-cookie in the credentials.

```
basic-credentials  = "Basic" SP basic-cookie
basic-cookie       = <base64 encoding of userid-password>
userid-password    = [ token ] ":" *text
```

There are no optional authentication parameters for the basic scheme. For example, if the user agent wishes to send the user-ID "Aladdin" and password "open sesame", it would use the following header field:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

The basic authentication scheme is a non-secure method of filtering unauthorized access to resources on an HTTP server. It is based on the assumption that the connection between the client and the server can be regarded as a trusted carrier. As this is not generally true on an open network, the basic authentication scheme should be used accordingly. In spite of this, clients are encouraged to implement the scheme in order to communicate with servers that use it.

11. Security Considerations

This section is meant to inform application developers, information providers, and users of the security limitations in HTTP/1.0 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.

11.1 Authentication of Clients

As mentioned in Section 10.1, the Basic authentication scheme is not a secure method of user authentication, nor does it prevent the Entity-Body from being transmitted in clear text across the physical network used as the carrier. HTTP/1.0 does not prevent additional authentication schemes and encryption mechanisms to be employed to increase security.

11.2 Idempotent Methods

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

In particular, the convention has been established that the GET and HEAD methods should never have the significance of taking an action other than retrieval. These methods should be considered "safe" and should not have side effects. This allows the client software to represent other methods, such as POST, PUT and DELETE, in a special way, so that the user is aware of the fact that an non-idempotent action is being requested.

Naturally, it is not possible to ensure that the server does not generate side-effects as a result of performing a GET request; in fact, some dynamic resources consider that a feature. The important distinction here is that the user did not request the side-effects, so therefore cannot be held accountable for them.

11.3 Abuse of Server Log Information

A server is in the position to save personal data about a user's requests which may identify their reading patterns or subjects of interest. This information is clearly confidential in nature and its handling may be constrained by law in certain countries. People using the HTTP protocol to provide data are responsible for ensuring that such material is not distributed without the permission of any individuals that are identifiable by the published results.

11.4 Transfer of Sensitive Information

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

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

Proxies which serve as a gateway through a network firewall should take special precautions regarding the transfer of header information that identifies the hosts behind the firewall. In particular, they should remove, or replace with sanitized versions, any Forwarded fields generated behind the firewall.

The Referer field allows reading patterns to be studied and reverse links drawn. Although it can be very useful, its power can be abused if user details are not separated from the information contained in the Referer. Even when the personal information has been removed, the Referer field may indicate a private document's URI whose publication would be inappropriate.

The information sent in the From field might conflict with the user's privacy interests or their site's security policy, and hence it should not be transmitted without the user being able to disable, enable, and modify the contents of the field. The user must be able to set the contents of this field within a user preference or application defaults configuration.

We suggest, though do not require, that a convenient toggle interface be provided for the user to enable or disable the sending of From and Referer information.

12. Acknowledgments

This specification makes heavy use of the augmented BNF and generic constructs defined by David H. Crocker for RFC 822 [8]. Similarly, it reuses many of the definitions provided by Nathaniel Borenstein and Ned Freed for MIME [6]. We hope that their inclusion in this specification will help reduce past confusion over the relationship between HTTP/1.0 and Internet mail message formats.

The HTTP protocol has evolved considerably over the past three years. It has benefited from a large and active developer community—the many people who have participated on the *www-talk* mailing list—and it is that community which has been most responsible for the success of HTTP and of the World-Wide Web in general. Marc Andreessen, Robert Cailliau, Daniel W. Connolly, Bob Denny, Jean Francois-Groff, Phillip M. Hallam-Baker, Håkon W. Lie, Ari Luotonen, Rob McCool, Dave Raggett, Tony Sanders, and Marc VanHeyningen deserve special recognition for their efforts in defining aspects of the protocol for early versions of this specification.

This document has benefited greatly from the comments of all those participating in the HTTP-WG. In addition to those already mentioned, the following individuals have contributed to this specification:

```
Gary Adams              Harald Tveit Alvestrand
Keith Ball              Brian Behlendorf
Paul Burchard           Maurizio Codogno
Mike Cowlishaw          Roman Czyborra
Michael A. Dolan        John Franks
Marc Hedlund            Koen Holtman
Alex Hopmann            Bob Jernigan
Shel Kaphan             Martijn Koster
Dave Kristol            Daniel LaLiberte
Albert Lunde            John C. Mallery
Larry Masinter          Mitra
Gavin Nicol             Bill Perry
Jeffrey Perry           Owen Rees
David Robinson          Marc Salomon
Rich Salz               Jim Seidman
Chuck Shotton           Eric W. Sink
Simon E. Spero          Robert S. Thau
François Yergeau        Mary Ellen Zurko
```

13. References

[1] H. Alvestrand. “Tags for the identification of languages.” RFC 1766, UNINETT, March 1995.

[2] F. Anklesaria, M. McCahill, P. Lindner, D. Johnson, D. Torrey, and B. Alberti. “The Internet Gopher Protocol: A distributed document search and retrieval protocol.” RFC 1436, University of Minnesota, March 1993.

[3] T. Berners-Lee. “Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web.” RFC 1630, CERN, June 1994.

[4] T. Berners-Lee and D. Connolly. “HyperText Markup Language Specification – 2.0.” Work in Progress (draft-ietf-html-spec-04.txt), MIT/W3C, June 1995.

[5] T. Berners-Lee, L. Masinter, and M. McCahill. “Uniform Resource Locators (URL).” RFC 1738, CERN, Xerox PARC, University of Minnesota, October 1994.

[6] N. Borenstein and N. Freed. “MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies.” RFC 1521, Bellcore, Innosoft, September 1993.

[7] R. Braden. “Requirements for Internet hosts - application and support.” STD 3, RFC 1123, IETF, October 1989.

[8] D. H. Crocker. “Standard for the Format of ARPA Internet Text Messages.” STD 11, RFC 822, UDEL, August 1982.

[9] F. Davis, B. Kahle, H. Morris, J. Salem, T. Shen, R. Wang, J. Sui, and M. Grinbaum. “WAIS Interface Protocol Prototype Functional Specification.” (v1.5), Thinking Machines Corporation, April 1990.

[10] R. Fielding. “Relative Uniform Resource Locators.” RFC 1808, UC Irvine, June 1995.

[11] M. Horton and R. Adams. “Standard for interchange of USENET messages.” RFC 1036 (Obsoletes RFC 850), AT&T Bell Laboratories, Center for Seismic Studies, December 1987.

[12] B. Kantor and P. Lapsley. “Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News.” RFC 977, UC San Diego, UC Berkeley, February 1986.

[13] K. Moore. “MIME (Multipurpose Internet Mail Extensions) Part Two: Message Header Extensions for Non-ASCII Text.” RFC 1522, University of Tennessee, September 1993.

[14] J. Postel. “Simple Mail Transfer Protocol.” STD 10, RFC 821, USC/ISI, August 1982.

[15] J. Postel. “Media Type Registration Procedure.” RFC 1590, USC/ISI, March 1994.

[16] J. Postel and J. K. Reynolds. “File Transfer Protocol (FTP).” STD 9, RFC 959, USC/ISI, October 1985.

[17] J. Reynolds and J. Postel. “Assigned Numbers.” STD 2, RFC 1700, USC/ISI, October 1994.

[18] K. Sollins and L. Masinter. "Functional Requirements for Uniform Resource Names." RFC 1737, MIT/LCS, Xerox Corporation, December 1994.

[19] US-ASCII. Coded Character Set – 7-Bit American Standard Code for Information Interchange. Standard ANSI X3.4-1986, ANSI, 1986.

[20] ISO-8859. International Standard -- Information Processing -- 8-bit Single-Byte Coded Graphic Character Sets -- Part 1: Latin Alphabet No. 1, ISO 8859-1:1987. Part 2: Latin alphabet No. 2, ISO 8859-2, 1987. Part 3: Latin alphabet No. 3, ISO 8859-3, 1988. Part 4: Latin alphabet No. 4, ISO 8859-4, 1988. Part 5: Latin/Cyrillic alphabet, ISO 8859-5, 1988. Part 6: Latin/Arabic alphabet, ISO 8859-6, 1987. Part 7: Latin/Greek alphabet, ISO 8859-7, 1987. Part 8: Latin/Hebrew alphabet, ISO 8859-8, 1988. Part 9: Latin alphabet No. 5, ISO 8859-9, 1990.

14. Authors' Addresses

Tim Berners-Lee
Director, W3 Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, U.S.A.
Tel: +1 (617) 253 5702
Fax: +1 (617) 258 8682
Email: timbl@w3.org

Roy T. Fielding
Department of Information and Computer Science
University of California
Irvine, CA 92717-3425, U.S.A.
Tel: +1 (714) 824-4049
Fax: +1 (714) 824-4056
Email: fielding@ics.uci.edu

Henrik Frystyk Nielsen
W3 Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, U.S.A.
Tel: +1 (617) 258 8143
Fax: +1 (617) 258 8682
Email: frystyk@w3.org

Appendices

These appendices are provided for informational reasons only — they do not form a part of the HTTP/1.0 specification.

A. Internet Media Type message/http

In addition to defining the HTTP/1.0 protocol, this document serves as the specification for the Internet media type "message/http". The following is to be registered with IANA [15].

```
Media Type name:         message
Media subtype name:      http
Required parameters:     none
Optional parameters:     version, msgtype
      version:  The HTTP-Version number of the enclosed message
                (e.g., "1.0"). If not present, the version can be
                determined from the first line of the body.
      msgtype:  The message type -- "request" or "response". If not
                present, the type can be determined from the first
                line of the body.
Encoding considerations: only "7bit", "8bit", or "binary" are
                         permitted
Security considerations: none
```

B. Tolerant Applications

Although this document specifies the requirements for the generation of HTTP/1.0 messages, not all applications will be correct in their implementation. We therefore recommend that operational applications be tolerant of deviations whenever those deviations can be interpreted unambiguously.

Clients should be tolerant in parsing the StatusLine and servers tolerant when parsing the RequestLine. In particular, they should accept any amount of SP or HT characters between fields, even though only a single SP is required.

The line terminator for HTTP-header fields is the sequence CRLF. However, we recommend that applications, when parsing such headers, recognize a single LF as a line terminator and ignore the leading CR.

C. Relationship to MIME

HTTP/1.0 reuses many of the constructs defined for Internet Mail (RFC 822 [8]) and the Multipurpose Internet Mail Extensions (MIME [6]) to allow entities to be transmitted in an open variety of representations and with extensible mechanisms. However, HTTP is not a MIME-conforming application. HTTP's performance requirements differ substantially from those of Internet mail. Since it is not limited by the restrictions of existing mail protocols and gateways, HTTP does not obey some of the constraints imposed by RFC 822 and MIME for mail transport.

This appendix describes specific areas where HTTP differs from MIME. Gateways to MIME-compliant protocols must be aware of these differences and provide the appropriate conversions where necessary. No conversion should be necessary for a MIME-conforming entity to be transferred using HTTP.

C.1 Conversion to Canonical Form

MIME requires that an entity be converted to canonical form prior to being transferred, as described in Appendix G of RFC 1521 [6]. Although HTTP does require media types to be transferred in canonical form, it changes the definition of "canonical form" for text-based media types as described in Section 3.4.1.

C.1.1 Representation of Line Breaks

MIME requires that the canonical form of any text type represent line breaks as CRLF and forbids the use of CR or LF outside of line break sequences. Since HTTP allows CRLF, bare CR, and bare LF (or the octet sequence(s) to which they would be translated for the given character set encoding) to indicate a line break within text content, recipients of an HTTP message cannot rely upon receiving MIME-canonical line breaks in text.

Where it is possible, a gateway from HTTP to a MIME-conformant protocol should translate all line breaks within text/* media types to the MIME canonical form of CRLF. However, this may be complicated by the presence of a Content-Encoding and by the fact that HTTP allows the use of some character set encodings which do not use octets 13 and 10 to represent CR and LF, as is the case for some multi-byte character set encodings.

C.1.2 Default Character Set Encoding

MIME requires that all subtypes of the top-level Content-Type "text" have a default character set encoding of US-ASCII [19]. In contrast, HTTP defines the default character set encoding for "text" to be ISO-8859-1 [20] (a superset of US-ASCII). Therefore, if a text/* media type given in the Content-Type header field does not already include an explicit charset parameter, the parameter

```
;charset="iso-8859-1"
```

should be added by the gateway if the entity contains any octets greater than 127.

C.2 Default Content-Transfer-Encoding

The default Content-Transfer-Encoding (CTE) for all MIME messages is "7bit". In contrast, HTTP defines the default CTE to be "binary". Therefore, if an entity does not include an explicit CTE header field, the gateway should apply either the "quoted-printable" or "base64" transfer encodings and add the appropriate Content-Transfer-Encoding field. At a minimum, the explicit CTE field of

```
Content-Transfer-Encoding: binary
```

should be added by the gateway if it is unwilling to apply a mail-safe transfer encoding.

C.3 Introduction of Content-Encoding

MIME does not include any concept equivalent to HTTP's Content-Encoding header field. Since this acts as a modifier on the media type, gateways to MIME-conformant protocols should either change the value of the Content-Type header field or decode the Entity-Body before forwarding the message.

> ***Note:*** *Some experimental applications of Content-Type for Internet mail have used a media-type parameter of ";conversions=<encoding-mechanisms>" to perform an equivalent function as Content-Encoding. However, this parameter is not part of the MIME specification at the time of this writing.*