HTTPbis Working Group	R. Fielding, Ed.
Internet-Draft	Adobe
Updates: 2817 (if approved)	J. Gettys
Obsoletes: 2145,2616 (if approved)	Alcatel-Lucent
Intended status: Standards Track	J. Mogul
Expires: September 15, 2011	HP
	H. Frystyk
	Microsoft
	L. Masinter
	Adobe
	P. Leach
	Microsoft
	T. Berners-Lee
	W3C/MIT
	Y. Lafon, Ed.
	W3C
	J. F. Reschke, Ed.
	greenbytes
	March 14, 2011

HTTP/1.1, part 1: URIs, Connections, and Message Parsing
draft-ietf-httpbis-p1-messaging-13

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypertext information systems. HTTP has been in use by the World Wide Web global information initiative since 1990. This document is Part 1 of the seven-part specification that defines the protocol referred to as "HTTP/1.1" and, taken together, obsoletes RFC 2616. Part 1 provides an overview of HTTP and its associated terminology, defines the "http" and "https" Uniform Resource Identifier (URI) schemes, defines the generic message syntax and parsing requirements for HTTP message frames, and describes general security concerns for implementations.

Editorial Note (To be removed by RFC Editor)

Discussion of this draft should take place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org). The current issues list is at http://tools.ietf.org/wg/httpbis/trac/report/3 and related documents (including fancy diffs) can be found at http://tools.ietf.org/wg/httpbis/.
The changes in this draft are summarized in Appendix Appendix D.14.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

The Hypertext Transfer Protocol (HTTP) is an application-level request/response protocol that uses extensible semantics and MIME-like message payloads for flexible interaction with network-based hypertext information systems. HTTP relies upon the Uniform Resource Identifier (URI) standard [RFC3986] to indicate the target resource and relationships between resources. Messages are passed in a format similar to that used by Internet mail [RFC5322] and the Multipurpose Internet Mail Extensions (MIME) [RFC2045] (see Appendix A of [Part3] for the differences between HTTP and MIME messages).

HTTP is a generic interface protocol for information systems. It is designed to hide the details of how a service is implemented by presenting a uniform interface to clients that is independent of the types of resources provided. Likewise, servers do not need to be aware of each client's purpose: an HTTP request can be considered in isolation rather than being associated with a specific type of client or a predetermined sequence of application steps. The result is a protocol that can be used effectively in many different contexts and for which implementations can evolve independently over time.

HTTP is also designed for use as an intermediation protocol for translating communication to and from non-HTTP information systems. HTTP proxies and gateways can provide access to alternative information services by translating their diverse protocols into a hypertext format that can be viewed and manipulated by clients in the same way as HTTP services.

One consequence of HTTP flexibility is that the protocol cannot be defined in terms of what occurs behind the interface. Instead, we are limited to defining the syntax of communication, the intent of received communication, and the expected behavior of recipients. If the communication is considered in isolation, then successful actions ought to be reflected in corresponding changes to the observable interface provided by servers. However, since multiple clients might act in parallel and perhaps at cross-purposes, we cannot require that such changes be observable beyond the scope of a single response.

This document is Part 1 of the seven-part specification of HTTP, defining the protocol referred to as "HTTP/1.1", obsoleting [RFC2616] and [RFC2145]. Part 1 describes the architectural elements that are used or referred to in HTTP, defines the "http" and "https" URI schemes, describes overall network operation and connection management, and defines HTTP message framing and forwarding requirements. Our goal is to define all of the mechanisms necessary for HTTP message handling that are independent of message semantics, thereby defining the complete set of requirements for message parsers and message-forwarding intermediaries.

1.1. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].
An implementation is not compliant if it fails to satisfy one or more of the "MUST" or "REQUIRED" level requirements for the protocols it implements. An implementation that satisfies all the "MUST" or "REQUIRED" level and all the "SHOULD" level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the "MUST" level requirements but not all the "SHOULD" level requirements for its protocols is said to be "conditionally compliant".

1.2. Syntax Notation

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234].
The following core rules are included by reference, as defined in [RFC5234], Appendix B.1: ALPHA (letters), CR (carriage return), CRLF (CR LF), CTL (controls), DIGIT (decimal 0-9), DQUOTE (double quote), HEXDIG (hexadecimal 0-9/A-F/a-f), LF (line feed), OCTET (any 8-bit sequence of data), SP (space), VCHAR (any visible [USASCII] character), and WSP (whitespace).
As a syntactic convention, ABNF rule names prefixed with "obs-" denote "obsolete" grammar rules that appear for historical reasons.

1.2.1. ABNF Extension: #rule

The #rule extension to the ABNF rules of [RFC5234] is used to improve readability.
A construct "#" is defined, similar to "*", for defining comma-delimited lists of elements. The full form is "<n>#<m>element" indicating at least <n> and at most <m> elements, each separated by a single comma (",") and optional whitespace (OWS, Section 1.2.2).
Thus,

```
  1#element => element *( OWS "," OWS element )
```

and:

```
  #element => [ 1#element ]
```

and for n >= 1 and m > 1:

```
  <n>#<m>element => element <n-1>*<m-1>( OWS "," OWS element )
```

For compatibility with legacy list rules, recipients SHOULD accept empty list elements. In other words, consumers would follow the list productions:

```
  #element => [ ( "," / element ) *( OWS "," [ OWS element ] ) ]

  1#element => *( "," OWS ) element *( OWS "," [ OWS element ] )
```

Note that empty elements do not contribute to the count of elements present, though.
For example, given these ABNF productions:

```
  example-list      = 1#example-list-elmt
  example-list-elmt = token ; see Section 1.2.2
```

Then these are valid values for example-list (not including the double quotes, which are present for delimitation only):

```
  "foo,bar"
  " foo ,bar,"
  "  foo , ,bar,charlie   "
  "foo ,bar,   charlie "
```

But these values would be invalid, as at least one non-empty element is required:

```
  ""
  ","
  ",   ,"
```

Appendix Appendix C shows the collected ABNF, with the list rules expanded as explained above.

1.2.2. Basic Rules

HTTP/1.1 defines the sequence CR LF as the end-of-line marker for all protocol elements other than the message-body (see Appendix Appendix A for tolerant applications).
This specification uses three rules to denote the use of linear whitespace: OWS (optional whitespace), RWS (required whitespace), and BWS ("bad" whitespace).
The OWS rule is used where zero or more linear whitespace octets might appear. OWS SHOULD either not be produced or be produced as a single SP. Multiple OWS octets that occur within field-content SHOULD be

replaced with a single SP before interpreting the field value or forwarding the message downstream.
RWS is used when at least one linear whitespace octet is required to separate field tokens. RWS SHOULD be produced as a single SP. Multiple RWS octets that occur within field-content SHOULD be replaced with a single SP before interpreting the field value or forwarding the message downstream.
BWS is used where the grammar allows optional whitespace for historical reasons but senders SHOULD NOT produce it in messages. HTTP/1.1 recipients MUST accept such bad optional whitespace and remove it before interpreting the field value or forwarding the message downstream.

```
  OWS            = *( [ obs-fold ] WSP )
                 ; "optional" whitespace
  RWS            = 1*( [ obs-fold ] WSP )
                 ; "required" whitespace
  BWS            = OWS
                 ; "bad" whitespace
  obs-fold       = CRLF
                 ; see Section 3.2
```

Many HTTP/1.1 header field values consist of words (token or quoted-string) separated by whitespace or special characters. These special characters MUST be in a quoted string to be used within a parameter value (as defined in Section 6.2).

```
  word           = token / quoted-string

  token          = 1*tchar

  tchar          = "!" / "#" / "$" / "%" / "&" / "'" / "*"
                 / "+" / "-" / "." / "^" / "_" / "`" / "|" / "~"
                 / DIGIT / ALPHA
                 ; any VCHAR, except special

  special        = "(" / ")" / "<" / ">" / "@" / ","
                 / ";" / ":" / "\" / DQUOTE / "/" / "["
                 / "]" / "?" / "=" / "{" / "}"
```

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

```
  quoted-string  = DQUOTE *( qdtext / quoted-pair ) DQUOTE
  qdtext         = OWS / %x21 / %x23-5B / %x5D-7E / obs-text
                 ; OWS / <VCHAR except DQUOTE and "\"> / obs-text
  obs-text       = %x80-FF
```

The backslash octet ("\") can be used as a single-octet quoting mechanism within quoted-string constructs:

```
  quoted-pair    = "\" ( WSP / VCHAR / obs-text )
```

Senders SHOULD NOT escape octets that do not require escaping (i.e., other than DQUOTE and the backslash octet).

2. HTTP-related architecture

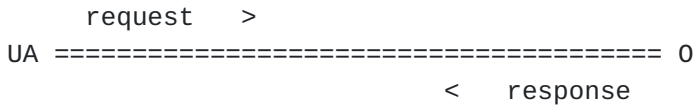
HTTP was created for the World Wide Web architecture and has evolved over time to support the scalability needs of a worldwide hypertext system. Much of that architecture is reflected in the terminology and syntax productions used to define HTTP.

2.1. Client/Server Messaging

HTTP is a stateless request/response protocol that operates by exchanging messages across a reliable transport or session-layer connection. An HTTP "client" is a program that establishes a connection to a server for the purpose of sending one or more HTTP requests. An HTTP "server" is a program that accepts connections in order to service HTTP requests by sending HTTP responses.

Note that the terms client and server refer only to the roles that these programs perform for a particular connection. The same program might act as a client on some connections and a server on others. We use the term "user agent" to refer to the program that initiates a request, such as a WWW browser, editor, or spider (web-traversing robot), and the term "origin server" to refer to the program that can originate authoritative responses to a request. For general requirements, we use the term "sender" to refer to whichever component sent a given message and the term "recipient" to refer to any component that receives the message.

Most HTTP communication consists of a retrieval request (GET) for a representation of some resource identified by a URI. In the simplest case, this might be accomplished via a single bidirectional connection (===) between the user agent (UA) and the origin server (O).



A client sends an HTTP request to the server in the form of a request message (Section 4), beginning with a method, URI, and protocol version, followed by MIME-like header fields containing request modifiers, client information, and payload metadata, an empty line to indicate the end of the header section, and finally the payload body (if any).

A server responds to the client's request by sending an HTTP response message (Section 5), beginning with a status line that includes the protocol version, a success or error code, and textual reason phrase, followed by MIME-like header fields containing server information,

resource metadata, and payload metadata, an empty line to indicate the end of the header section, and finally the payload body (if any).
The following example illustrates a typical message exchange for a GET request on the URI "http://www.example.com/hello.txt":
client request:

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
Host: www.example.com
Accept: */*
```

server response:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 14
Vary: Accept-Encoding
Content-Type: text/plain

Hello World!
```

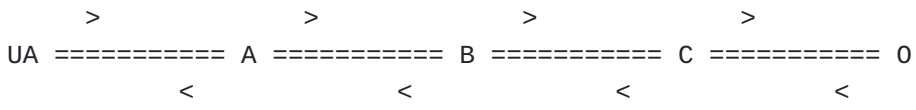
2.2. Connections and Transport Independence

HTTP messaging is independent of the underlying transport or session-layer connection protocol(s). HTTP only presumes a reliable transport with in-order delivery of requests and the corresponding in-order delivery of responses. The mapping of HTTP request and response structures onto the data units of the underlying transport protocol is outside the scope of this specification.
The specific connection protocols to be used for an interaction are determined by client configuration and the target resource's URI. For example, the "http" URI scheme (Section 2.6.1) indicates a default connection of TCP over IP, with a default TCP port of 80, but the client might be configured to use a proxy via some other connection port or protocol instead of using the defaults.
A connection might be used for multiple HTTP request/response exchanges, as defined in Section 7.1.

2.3. Intermediaries

HTTP enables the use of intermediaries to satisfy requests through a chain of connections. There are three common forms of HTTP intermediary: proxy, gateway, and tunnel. In some cases, a single

intermediary might act as an origin server, proxy, gateway, or tunnel, switching behavior based on the nature of each request.



The figure above shows three intermediaries (A, B, and C) between the user agent and origin server. A request or response message that travels the whole chain will pass through four separate connections. Some HTTP communication options might apply only to the connection with the nearest, non-tunnel neighbor, only to the end-points of the chain, or to all connections along the chain. Although the diagram is linear, each participant might be engaged in multiple, simultaneous communications. For example, B might be receiving requests from many clients other than A, and/or forwarding requests to servers other than C, at the same time that it is handling A's request.

We use the terms "upstream" and "downstream" to describe various requirements in relation to the directional flow of a message: all messages flow from upstream to downstream. Likewise, we use the terms "inbound" and "outbound" to refer to directions in relation to the request path: "inbound" means toward the origin server and "outbound" means toward the user agent.

A "proxy" is a message forwarding agent that is selected by the client, usually via local configuration rules, to receive requests for some type(s) of absolute URI and attempt to satisfy those requests via translation through the HTTP interface. Some translations are minimal, such as for proxy requests for "http" URIs, whereas other requests might require translation to and from entirely different application-layer protocols. Proxies are often used to group an organization's HTTP requests through a common intermediary for the sake of security, annotation services, or shared caching.

An HTTP-to-HTTP proxy is called a "transforming proxy" if it is designed or configured to modify request or response messages in a semantically meaningful way (i.e., modifications, beyond those required by normal HTTP processing, that change the message in a way that would be significant to the original sender or potentially significant to downstream recipients). For example, a transforming proxy might be acting as a shared annotation server (modifying responses to include references to a local annotation database), a malware filter, a format transcoder, or an intranet-to-Internet privacy filter. Such transformations are presumed to be desired by the client (or client organization) that selected the proxy and are beyond the scope of this specification. However, when a proxy is not intended to transform a given message, we use the term "non-transforming proxy" to target requirements that preserve HTTP message semantics.

A "gateway" (a.k.a., "reverse proxy") is a receiving agent that acts as a layer above some other server(s) and translates the received requests to the underlying server's protocol. Gateways are often used

to encapsulate legacy or untrusted information services, to improve server performance through "accelerator" caching, and to enable partitioning or load-balancing of HTTP services across multiple machines.

A gateway behaves as an origin server on its outbound connection and as a user agent on its inbound connection. All HTTP requirements applicable to an origin server also apply to the outbound communication of a gateway. A gateway communicates with inbound servers using any protocol that it desires, including private extensions to HTTP that are outside the scope of this specification. However, an HTTP-to-HTTP gateway that wishes to interoperate with third-party HTTP servers MUST comply with HTTP user agent requirements on the gateway's inbound connection and MUST implement the Connection (Section 9.1) and Via (Section 9.9) header fields for both connections.

A "tunnel" acts as a blind relay between two connections without changing the messages. Once active, a tunnel is not considered a party to the HTTP communication, though the tunnel might have been initiated by an HTTP request. A tunnel ceases to exist when both ends of the relayed connection are closed. Tunnels are used to extend a virtual connection through an intermediary, such as when transport-layer security is used to establish private communication through a shared firewall proxy.

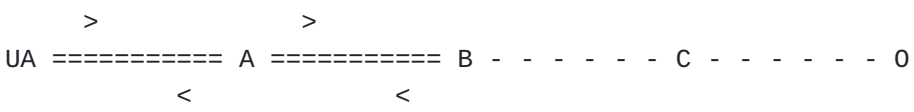
In addition, there may exist network intermediaries that are not considered part of the HTTP communication but nevertheless act as filters or redirecting agents (usually violating HTTP semantics, causing security problems, and otherwise making a mess of things). Such a network intermediary, often referred to as an "interception proxy" [RFC3040], "transparent proxy" [RFC1919], or "captive portal", differs from an HTTP proxy because it has not been selected by the client. Instead, the network intermediary redirects outgoing TCP port 80 packets (and occasionally other common port traffic) to an internal HTTP server. Interception proxies are commonly found on public network access points, as a means of enforcing account subscription prior to allowing use of non-local Internet services, and within corporate firewalls to enforce network usage policies. They are indistinguishable from a man-in-the-middle attack.

2.4. Caches

A "cache" is a local store of previous response messages and the subsystem that controls its message storage, retrieval, and deletion. A cache stores cacheable responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests. Any client or server MAY employ a cache, though a cache cannot be used by a server while it is acting as a tunnel.

The effect of a cache is that the request/response chain is shortened if one of the participants along the chain has a cached response applicable to that request. The following illustrates the resulting

chain if B has a cached copy of an earlier response from O (via C) for a request which has not been cached by UA or A.



A response is "cacheable" if a cache is allowed to store a copy of the response message for use in answering subsequent requests. Even when a response is cacheable, there might be additional constraints placed by the client or by the origin server on when that cached response can be used for a particular request. HTTP requirements for cache behavior and cacheable responses are defined in Section 2 of [Part6].
There are a wide variety of architectures and configurations of caches and proxies deployed across the World Wide Web and inside large organizations. These systems include national hierarchies of proxy caches to save transoceanic bandwidth, systems that broadcast or multicast cache entries, organizations that distribute subsets of cached data via optical media, and so on.

2.5. Protocol Versioning

HTTP uses a "<major>.<minor>" numbering scheme to indicate versions of the protocol. This specification defines version "1.1". The protocol version as a whole indicates the sender's compliance with the set of requirements laid out in that version's corresponding specification of HTTP.
The version of an HTTP message is indicated by an HTTP-Version field in the first line of the message. HTTP-Version is case-sensitive.

```
HTTP-Version   = HTTP-Prot-Name "/" 1*DIGIT "." 1*DIGIT
HTTP-Prot-Name = %x48.54.54.50 ; "HTTP", case-sensitive
```

The HTTP version number consists of two non-negative decimal integers separated by a "." (period or decimal point). The first number ("major version") indicates the HTTP messaging syntax, whereas the second number ("minor version") indicates the highest minor version to which the sender is at least conditionally compliant and able to understand for future communication. The minor version advertises the sender's communication capabilities even when the sender is only using a backwards-compatible subset of the protocol, thereby letting the recipient know that more advanced features can be used in response (by servers) or in future requests (by clients).
When comparing HTTP versions, the numbers MUST be compared numerically rather than lexically. For example, HTTP/2.4 is a lower version than HTTP/2.13, which in turn is lower than HTTP/12.3. Leading zeros MUST be ignored by recipients and MUST NOT be sent.
When an HTTP/1.1 message is sent to an HTTP/1.0 recipient [RFC1945] or a recipient whose version is unknown, the HTTP/1.1 message is

constructed such that it can be interpreted as a valid HTTP/1.0 message if all of the newer features are ignored. This specification places recipient-version requirements on some new features so that a compliant sender will only use compatible features until it has determined, through configuration or the receipt of a message, that the recipient supports HTTP/1.1.

The interpretation of an HTTP header field does not change between minor versions of the same major version, though the default behavior of a recipient in the absence of such a field can change. Unless specified otherwise, header fields defined in HTTP/1.1 are defined for all versions of HTTP/1.x. In particular, the Host and Connection header fields ought to be implemented by all HTTP/1.x implementations whether or not they advertise compliance with HTTP/1.1.

New header fields can be defined such that, when they are understood by a recipient, they might override or enhance the interpretation of previously defined header fields. When an implementation receives an unrecognized header field, the recipient MUST ignore that header field for local processing regardless of the message's HTTP version. An unrecognized header field received by a proxy MUST be forwarded downstream unless the header field's field-name is listed in the message's Connection header-field (see Section 9.1). These requirements allow HTTP's functionality to be enhanced without requiring prior update of all compliant intermediaries.

Intermediaries that process HTTP messages (i.e., all intermediaries other than those acting as a tunnel) MUST send their own HTTP-Version in forwarded messages. In other words, they MUST NOT blindly forward the first line of an HTTP message without ensuring that the protocol version matches what the intermediary understands, and is at least conditionally compliant to, for both the receiving and sending of messages. Forwarding an HTTP message without rewriting the HTTP-Version might result in communication errors when downstream recipients use the message sender's version to determine what features are safe to use for later communication with that sender.

An HTTP client SHOULD send a request version equal to the highest version for which the client is at least conditionally compliant and whose major version is no higher than the highest version supported by the server, if this is known. An HTTP client MUST NOT send a version for which it is not at least conditionally compliant.

An HTTP client MAY send a lower request version if it is known that the server incorrectly implements the HTTP specification, but only after the client has attempted at least one normal request and determined from the response status or header fields (e.g., Server) that the server improperly handles higher request versions.

An HTTP server SHOULD send a response version equal to the highest version for which the server is at least conditionally compliant and whose major version is less than or equal to the one received in the request. An HTTP server MUST NOT send a version for which it is not at least conditionally compliant. A server MAY send a 505 (HTTP Version

Not Supported) response if it cannot send a response using the major version used in the client's request.
An HTTP server MAY send an HTTP/1.0 response to an HTTP/1.0 request if it is known or suspected that the client incorrectly implements the HTTP specification and is incapable of correctly processing later version responses, such as when a client fails to parse the version number correctly or when an intermediary is known to blindly forward the HTTP-Version even when it doesn't comply with the given minor version of the protocol. Such protocol downgrades SHOULD NOT be performed unless triggered by specific client attributes, such as when one or more of the request header fields (e.g., User-Agent) uniquely match the values sent by a client known to be in error.
The intention of HTTP's versioning design is that the major number will only be incremented if an incompatible message syntax is introduced, and that the minor number will only be incremented when changes made to the protocol have the effect of adding to the message semantics or implying additional capabilities of the sender. However, the minor version was not incremented for the changes introduced between [RFC2068] and [RFC2616], and this revision is specifically avoiding any such changes to the protocol.

2.6. Uniform Resource Identifiers

Uniform Resource Identifiers (URIs) [RFC3986] are used throughout HTTP as the means for identifying resources. URI references are used to target requests, indicate redirects, and define relationships. HTTP does not limit what a resource might be; it merely defines an interface that can be used to interact with a resource via HTTP. More information on the scope of URIs and resources can be found in [RFC3986].
This specification adopts the definitions of "URI-reference", "absolute-URI", "relative-part", "port", "host", "path-abempty", "path-absolute", "query", and "authority" from the URI generic syntax [RFC3986]. In addition, we define a partial-URI rule for protocol elements that allow a relative URI but not a fragment.

```
  URI-reference = <URI-reference, defined in [RFC3986], Section 4.1>
  absolute-URI  = <absolute-URI, defined in [RFC3986], Section 4.3>
  relative-part = <relative-part, defined in [RFC3986], Section 4.2>
  authority     = <authority, defined in [RFC3986], Section 3.2>
  path-abempty  = <path-abempty, defined in [RFC3986], Section 3.3>
  path-absolute = <path-absolute, defined in [RFC3986], Section 3.3>
  port          = <port, defined in [RFC3986], Section 3.2.3>
  query         = <query, defined in [RFC3986], Section 3.4>
  uri-host      = <host, defined in [RFC3986], Section 3.2.2>

  partial-URI   = relative-part [ "?" query ]
```

Each protocol element in HTTP that allows a URI reference will indicate in its ABNF production whether the element allows any form of reference

(URI-reference), only a URI in absolute form (absolute-URI), only the path and optional query components, or some combination of the above. Unless otherwise indicated, URI references are parsed relative to the effective request URI, which defines the default base URI for references in both the request and its corresponding response.

2.6.1. http URI scheme

The "http" URI scheme is hereby defined for the purpose of minting identifiers according to their association with the hierarchical namespace governed by a potential HTTP origin server listening for TCP connections on a given port.

```
http-URI = "http:" "//" authority path-abempty [ "?" query ]
```

The HTTP origin server is identified by the generic syntax's authority *[uri]* component, which includes a host identifier and optional TCP port ([RFC3986], Section 3.2.2). The remainder of the URI, consisting of both the hierarchical path component and optional query component, serves as an identifier for a potential resource within that origin server's name space.

If the host identifier is provided as an IP literal or IPv4 address, then the origin server is any listener on the indicated TCP port at that IP address. If host is a registered name, then that name is considered an indirect identifier and the recipient might use a name resolution service, such as DNS, to find the address of a listener for that host. The host MUST NOT be empty; if an "http" URI is received with an empty host, then it MUST be rejected as invalid. If the port subcomponent is empty or not given, then TCP port 80 is assumed (the default reserved port for WWW services).

Regardless of the form of host identifier, access to that host is not implied by the mere presence of its name or address. The host might or might not exist and, even when it does exist, might or might not be running an HTTP server or listening to the indicated port. The "http" URI scheme makes use of the delegated nature of Internet names and addresses to establish a naming authority (whatever entity has the ability to place an HTTP server at that Internet name or address) and allows that authority to determine which names are valid and how they might be used.

When an "http" URI is used within a context that calls for access to the indicated resource, a client MAY attempt access by resolving the host to an IP address, establishing a TCP connection to that address on the indicated port, and sending an HTTP request message to the server containing the URI's identifying data as described in Section 4. If the server responds to that request with a non-interim HTTP response message, as described in Section 5, then that response is considered an authoritative answer to the client's request.

Although HTTP is independent of the transport protocol, the "http" scheme is specific to TCP-based services because the name delegation

process depends on TCP for establishing authority. An HTTP service based on some other underlying connection protocol would presumably be identified using a different URI scheme, just as the "https" scheme (below) is used for servers that require an SSL/TLS transport layer on a connection. Other protocols might also be used to provide access to "http" identified resources — it is only the authoritative interface used for mapping the namespace that is specific to TCP.

The URI generic syntax for authority also includes a deprecated userinfo subcomponent ([RFC3986], Section 3.2.1) for including user authentication information in the URI. Some implementations make use of the userinfo component for internal configuration of authentication information, such as within command invocation options, configuration files, or bookmark lists, even though such usage might expose a user identifier or password. Senders MUST NOT include a userinfo subcomponent (and its "@" delimiter) when transmitting an "http" URI in a message. Recipients of HTTP messages that contain a URI reference SHOULD parse for the existence of userinfo and treat its presence as an error, likely indicating that the deprecated subcomponent is being used to obscure the authority for the sake of phishing attacks.

2.6.2. https URI scheme

The "https" URI scheme is hereby defined for the purpose of minting identifiers according to their association with the hierarchical namespace governed by a potential HTTP origin server listening for SSL/TLS-secured connections on a given TCP port.

All of the requirements listed above for the "http" scheme are also requirements for the "https" scheme, except that a default TCP port of 443 is assumed if the port subcomponent is empty or not given, and the TCP connection MUST be secured for privacy through the use of strong encryption prior to sending the first HTTP request.

```
  https-URI = "https:" "//" authority path-abempty [ "?" query ]
```

Unlike the "http" scheme, responses to "https" identified requests are never "public" and thus MUST NOT be reused for shared caching. They can, however, be reused in a private cache if the message is cacheable by default in HTTP or specifically indicated as such by the Cache-Control header field (Section 3.2 of [Part6]).

Resources made available via the "https" scheme have no shared identity with the "http" scheme even if their resource identifiers indicate the same authority (the same host listening to the same TCP port). They are distinct name spaces and are considered to be distinct origin servers. However, an extension to HTTP that is defined to apply to entire host domains, such as the Cookie protocol [draft-ietf-httpstate-cookie], can allow information set by one service to impact communication with other services within a matching group of host domains.

The process for authoritative access to an "https" identified resource is defined in [RFC2818].

2.6.3. http and https URI Normalization and Comparison

Since the "http" and "https" schemes conform to the URI generic syntax, such URIs are normalized and compared according to the algorithm defined in [RFC3986], Section 6, using the defaults described above for each scheme.
If the port is equal to the default port for a scheme, the normal form is to elide the port subcomponent. Likewise, an empty path component is equivalent to an absolute path of "/", so the normal form is to provide a path of "/" instead. The scheme and host are case-insensitive and normally provided in lowercase; all other components are compared in a case-sensitive manner. Characters other than those in the "reserved" set are equivalent to their percent-encoded octets (see [RFC3986], Section 2.1): the normal form is to not encode them.
For example, the following three URIs are equivalent:

```
http://example.com:80/~smith/home.html
http://EXAMPLE.com/%7Esmith/home.html
http://EXAMPLE.com:/%7esmith/home.html
```

3. Message Format

All HTTP/1.1 messages consist of a start-line followed by a sequence of octets in a format similar to the Internet Message Format [RFC5322]: zero or more header fields (collectively referred to as the "headers" or the "header section"), an empty line indicating the end of the header section, and an optional message-body.
An HTTP message can either be a request from client to server or a response from server to client. Syntactically, the two types of message differ only in the start-line, which is either a Request-Line (for requests) or a Status-Line (for responses), and in the algorithm for determining the length of the message-body (Section 3.3). In theory, a client could receive requests and a server could receive responses, distinguishing them by their different start-line formats, but in practice servers are implemented to only expect a request (a response is interpreted as an unknown or invalid request method) and clients are implemented to only expect a response.

```
  HTTP-message    = start-line
                    *( header-field CRLF )
                    CRLF
                    [ message-body ]
  start-line      = Request-Line / Status-Line
```

Implementations MUST NOT send whitespace between the start-line and the first header field. The presence of such whitespace in a request might be an attempt to trick a server into ignoring that field or processing the line after it as a new request, either of which might result in a security vulnerability if other implementations within the request

chain interpret the same message differently. Likewise, the presence of such whitespace in a response might be ignored by some clients or cause others to cease parsing.

3.1. Message Parsing Robustness

In the interest of robustness, servers SHOULD ignore at least one empty line received where a Request-Line is expected. In other words, if the server is reading the protocol stream at the beginning of a message and receives a CRLF first, it SHOULD ignore the CRLF.
Some old HTTP/1.0 client implementations send an extra CRLF after a POST request as a lame workaround for some early server applications that failed to read message-body content that was not terminated by a line-ending. An HTTP/1.1 client MUST NOT preface or follow a request with an extra CRLF. If terminating the request message-body with a line-ending is desired, then the client MUST include the terminating CRLF octets as part of the message-body length.
When a server listening only for HTTP request messages, or processing what appears from the start-line to be an HTTP request message, receives a sequence of octets that does not match the HTTP-message grammar aside from the robustness exceptions listed above, the server MUST respond with an HTTP/1.1 400 (Bad Request) response.
The normal procedure for parsing an HTTP message is to read the start-line into a structure, read each header field into a hash table by field name until the empty line, and then use the parsed data to determine if a message-body is expected. If a message-body has been indicated, then it is read as a stream until an amount of octets equal to the message-body length is read or the connection is closed. Care must be taken to parse an HTTP message as a sequence of octets in an encoding that is a superset of US-ASCII. Attempting to parse HTTP as a stream of Unicode characters in a character encoding like UTF-16 might introduce security flaws due to the differing ways that such parsers interpret invalid characters.
HTTP allows the set of defined header fields to be extended without changing the protocol version (see Section 10.1). Unrecognized header fields MUST be forwarded by a proxy unless the proxy is specifically configured to block or otherwise transform such fields. Unrecognized header fields SHOULD be ignored by other recipients.

3.2. Header Fields

Each HTTP header field consists of a case-insensitive field name followed by a colon (":"), optional whitespace, and the field value.

```
  header-field   = field-name ":" OWS [ field-value ] OWS
  field-name     = token
  field-value    = *( field-content / OWS )
  field-content  = *( WSP / VCHAR / obs-text )
```

No whitespace is allowed between the header field name and colon. For security reasons, any request message received containing such whitespace MUST be rejected with a response code of 400 (Bad Request). A proxy MUST remove any such whitespace from a response message before forwarding the message downstream.
A field value MAY be preceded by optional whitespace (OWS); a single SP is preferred. The field value does not include any leading or trailing white space: OWS occurring before the first non-whitespace octet of the field value or after the last non-whitespace octet of the field value is ignored and SHOULD be removed before further processing (as this does not change the meaning of the header field).
The order in which header fields with differing field names are received is not significant. However, it is "good practice" to send header fields that contain control data first, such as Host on requests and Date on responses, so that implementations can decide when not to handle a message as early as possible. A server MUST wait until the entire header section is received before interpreting a request message, since later header fields might include conditionals, authentication credentials, or deliberately misleading duplicate header fields that would impact request processing.
Multiple header fields with the same field name MUST NOT be sent in a message unless the entire field value for that header field is defined as a comma-separated list [i.e., #(values)]. Multiple header fields with the same field name can be combined into one "field-name: field-value" pair, without changing the semantics of the message, by appending each subsequent field value to the combined field value in order, separated by a comma. The order in which header fields with the same field name are received is therefore significant to the interpretation of the combined field value; a proxy MUST NOT change the order of these field values when forwarding a message.

> *Note: The "Set-Cookie" header field as implemented in practice can occur multiple times, but does not use the list syntax, and thus cannot be combined into a single line ([draft-ietf-httpstate-cookie]). (See Appendix A.2.3 of [Kri2001] for details.) Also note that the Set-Cookie2 header field specified in [RFC2965] does not share this problem.

Historically, HTTP header field values could be extended over multiple lines by preceding each extra line with at least one space or horizontal tab octet (line folding). This specification deprecates such line folding except within the message/http media type (Section 10.3.1). HTTP/1.1 senders MUST NOT produce messages that include line folding (i.e., that contain any field-content that matches the obs-fold rule) unless the message is intended for packaging within the message/http media type. HTTP/1.1 recipients SHOULD accept line folding and replace any embedded obs-fold whitespace with a single SP prior to interpreting the field value or forwarding the message downstream.

Historically, HTTP has allowed field content with text in the ISO-8859-1 [ISO-8859-1] character encoding and supported other character sets only through use of [RFC2047] encoding. In practice, most HTTP header field values use only a subset of the US-ASCII character encoding [USASCII]. Newly defined header fields SHOULD limit their field values to US-ASCII octets. Recipients SHOULD treat other (obs-text) octets in field content as opaque data.
Comments can be included in some HTTP header fields by surrounding the comment text with parentheses. Comments are only allowed in fields containing "comment" as part of their field value definition.

```
  comment        = "(" *( ctext / quoted-cpair / comment ) ")"
  ctext          = OWS / %x21-27 / %x2A-5B / %x5D-7E / obs-text
                 ; OWS / <VCHAR except "(", ")", and "\"> / obs-text
```

The backslash octet ("\") can be used as a single-octet quoting mechanism within comment constructs:

```
  quoted-cpair    = "\" ( WSP / VCHAR / obs-text )
```

Senders SHOULD NOT escape octets that do not require escaping (i.e., other than the backslash octet "\" and the parentheses "(" and ")").

3.3. Message Body

The message-body (if any) of an HTTP message is used to carry the payload body associated with the request or response.

```
  message-body = *OCTET
```

The message-body differs from the payload body only when a transfer-coding has been applied, as indicated by the Transfer-Encoding header field (Section 9.7). If more than one Transfer-Encoding header field is present in a message, the multiple field-values MUST be combined into one field-value, according to the algorithm defined in Section 3.2, before determining the message-body length.
When one or more transfer-codings are applied to a payload in order to form the message-body, the Transfer-Encoding header field MUST contain the list of transfer-codings applied. Transfer-Encoding is a property of the message, not of the payload, and thus MAY be added or removed by any implementation along the request/response chain under the constraints found in Section 6.2.
If a message is received that has multiple Content-Length header fields (Section 9.2) with field-values consisting of the same decimal value, or a single Content-Length header field with a field value containing a list of identical decimal values (e.g., "Content-Length: 42, 42"), indicating that duplicate Content-Length header fields have been generated or combined by an upstream message processor, then the recipient MUST replace the duplicated fields or field-values with a

single valid Content-Length field containing that decimal value prior to determining the message-body length.

The rules for when a message-body is allowed in a message differ for requests and responses.

The presence of a message-body in a request is signaled by the inclusion of a Content-Length or Transfer-Encoding header field in the request's header fields, even if the request method does not define any use for a message-body. This allows the request message framing algorithm to be independent of method semantics.

For response messages, whether or not a message-body is included with a message is dependent on both the request method and the response status code (Section 5.1.1). Responses to the HEAD request method never include a message-body because the associated response header fields (e.g., Transfer-Encoding, Content-Length, etc.) only indicate what their values would have been if the request method had been GET. All 1xx (Informational), 204 (No Content), and 304 (Not Modified) responses MUST NOT include a message-body. All other responses do include a message-body, although the body MAY be of zero length.

The length of the message-body is determined by one of the following (in order of precedence):

1. Any response to a HEAD request and any response with a status code of 100-199, 204, or 304 is always terminated by the first empty line after the header fields, regardless of the header fields present in the message, and thus cannot contain a message-body.

2. If a Transfer-Encoding header field is present and the "chunked" transfer-coding (Section 6.2) is the final encoding, the message-body length is determined by reading and decoding the chunked data until the transfer-coding indicates the data is complete.

   If a Transfer-Encoding header field is present in a response and the "chunked" transfer-coding is not the final encoding, the message-body length is determined by reading the connection until it is closed by the server. If a Transfer-Encoding header field is present in a request and the "chunked" transfer-coding is not the final encoding, the message-body length cannot be determined reliably; the server MUST respond with the 400 (Bad Request) status code and then close the connection.

   If a message is received with both a Transfer-Encoding header field and a Content-Length header field, the Transfer-Encoding overrides the Content-Length. Such a message might indicate an attempt to perform request or response smuggling (bypass of security-related checks on message routing or content) and thus ought to be handled as an error. The provided Content-Length MUST be removed, prior to forwarding the message downstream, or

replaced with the real message-body length after the transfer-coding is decoded.

3. If a message is received without Transfer-Encoding and with either multiple Content-Length header fields having differing field-values or a single Content-Length header field having an invalid value, then the message framing is invalid and MUST be treated as an error to prevent request or response smuggling. If this is a request message, the server MUST respond with a 400 (Bad Request) status code and then close the connection. If this is a response message received by a proxy, the proxy MUST discard the received response, send a 502 (Bad Gateway) status code as its downstream response, and then close the connection. If this is a response message received by a user-agent, it MUST be treated as an error by discarding the message and closing the connection.

4. If a valid Content-Length header field is present without Transfer-Encoding, its decimal value defines the message-body length in octets. If the actual number of octets sent in the message is less than the indicated Content-Length, the recipient MUST consider the message to be incomplete and treat the connection as no longer usable. If the actual number of octets sent in the message is more than the indicated Content-Length, the recipient MUST only process the message-body up to the field value's number of octets; the remainder of the message MUST either be discarded or treated as the next message in a pipeline. For the sake of robustness, a user-agent MAY attempt to detect and correct such an error in message framing if it is parsing the response to the last request on on a connection and the connection has been closed by the server.

5. If this is a request message and none of the above are true, then the message-body length is zero (no message-body is present).

6. Otherwise, this is a response message without a declared message-body length, so the message-body length is determined by the number of octets received prior to the server closing the connection.

Since there is no way to distinguish a successfully completed, close-delimited message from a partially-received message interrupted by network failure, implementations SHOULD use encoding or length-delimited messages whenever possible. The close-delimiting feature exists primarily for backwards compatibility with HTTP/1.0.
A server MAY reject a request that contains a message-body but not a Content-Length by responding with 411 (Length Required).

Unless a transfer-coding other than "chunked" has been applied, a client that sends a request containing a message-body SHOULD use a valid Content-Length header field if the message-body length is known in advance, rather than the "chunked" encoding, since some existing services respond to "chunked" with a 411 (Length Required) status code even though they understand the chunked encoding. This is typically because such services are implemented via a gateway that requires a content-length in advance of being called and the server is unable or unwilling to buffer the entire request before processing.

A client that sends a request containing a message-body MUST include a valid Content-Length header field if it does not know the server will handle HTTP/1.1 (or later) requests; such knowledge can be in the form of specific user configuration or by remembering the version of a prior received response.

Request messages that are prematurely terminated, possibly due to a cancelled connection or a server-imposed time-out exception, MUST result in closure of the connection; sending an HTTP/1.1 error response prior to closing the connection is OPTIONAL. Response messages that are prematurely terminated, usually by closure of the connection prior to receiving the expected number of octets or by failure to decode a transfer-encoded message-body, MUST be recorded as incomplete. A user agent MUST NOT render an incomplete response message-body as if it were complete (i.e., some indication must be given to the user that an error occurred). Cache requirements for incomplete responses are defined in Section 2.1.1 of [Part6].

A server MUST read the entire request message-body or close the connection after sending its response, since otherwise the remaining data on a persistent connection would be misinterpreted as the next request. Likewise, a client MUST read the entire response message-body if it intends to reuse the same connection for a subsequent request. Pipelining multiple requests on a connection is described in Section 7.1.2.2.

3.4. General 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 payload being transferred. These header fields apply only to the message being transmitted.

Header Field Name	Defined in...
Connection	Section 9.1
Date	Section 9.3
Trailer	Section 9.6
Transfer-Encoding	Section 9.7
Upgrade	Section 9.8

Header Field Name	Defined in...
Via	Section 9.9

4. Request

A request message from a client to a server begins with a Request-Line, followed by zero or more header fields, an empty line signifying the end of the header block, and an optional message body.

```
  Request       = Request-Line              ; Section 4.1
                  *( header-field CRLF )    ; Section 3.2
                  CRLF
                  [ message-body ]          ; Section 3.3
```

4.1. Request-Line

The Request-Line begins with a method token, followed by a single space (SP), the request-target, another single space (SP), the protocol version, and ending with CRLF.

```
  Request-Line   = Method SP request-target SP HTTP-Version CRLF
```

4.1.1. Method

The Method token indicates the request method to be performed on the target resource. The request method is case-sensitive.

```
  Method         = token
```

4.1.2. request-target

The request-target identifies the target resource upon which to apply the request. In most cases, the user agent is provided a URI reference from which it determines an absolute URI for identifying the target resource. When a request to the resource is initiated, all or part of that URI is used to construct the HTTP request-target.

```
  request-target = "*"
                 / absolute-URI
                 / ( path-absolute [ "?" query ] )
                 / authority
```

The four options for request-target are dependent on the nature of the request.
 The asterisk "*" form of request-target, which MUST NOT be used with any request method other than OPTIONS, means that the request applies to the server as a whole (the listening process) rather than to a specific named resource at that server. For example,

```
  OPTIONS * HTTP/1.1
```

The "absolute-URI" form is REQUIRED when the request is being made to a proxy. The proxy is requested to either forward the request or service it from a valid cache, and then return the response. Note that the proxy MAY forward the request on to another proxy or directly to the server specified by the absolute-URI. In order to avoid request loops, a proxy that forwards requests to other proxies MUST be able to recognize and exclude all of its own server names, including any aliases, local variations, and the numeric IP address. An example Request-Line would be:

```
  GET http://www.example.org/pub/WWW/TheProject.html HTTP/1.1
```

To allow for transition to absolute-URIs in all requests in future versions of HTTP, all HTTP/1.1 servers MUST accept the absolute-URI form in requests, even though HTTP/1.1 clients will only generate them in requests to proxies.

If a proxy receives a host name that is not a fully qualified domain name, it MAY add its domain to the host name it received. If a proxy receives a fully qualified domain name, the proxy MUST NOT change the host name.

The "authority form" is only used by the CONNECT request method (Section 7.9 of [Part2]).

The most common form of request-target is that used when making a request to an origin server ("origin form"). In this case, the absolute path and query components of the URI MUST be transmitted as the request-target, and the authority component MUST be transmitted in a Host header field. For example, a client wishing to retrieve a representation of the resource, as identified above, directly from the origin server would open (or reuse) a TCP connection to port 80 of the host "www.example.org" and send the lines:

```
  GET /pub/WWW/TheProject.html HTTP/1.1
  Host: www.example.org
```

followed by the remainder of the Request. Note that the origin form of request-target always starts with an absolute path; if the target resource's URI path is empty, then an absolute path of "/" MUST be provided in the request-target.

If a proxy receives an OPTIONS request with an absolute-URI form of request-target in which the URI has an empty path and no query component, then the last proxy on the request chain MUST use a request-target of "*" when it forwards the request to the indicated origin server.

For example, the request

```
OPTIONS http://www.example.org:8001 HTTP/1.1
```

would be forwarded by the final proxy as

```
OPTIONS * HTTP/1.1
Host: www.example.org:8001
```

after connecting to port 8001 of host "www.example.org".
The request-target is transmitted in the format specified in Section 2.6.1. If the request-target is percent-encoded ([RFC3986], Section 2.1), the origin server MUST decode the request-target in order to properly interpret the request. Servers SHOULD respond to invalid request-targets with an appropriate status code.
A non-transforming proxy MUST NOT rewrite the "path-absolute" part of the received request-target when forwarding it to the next inbound server, except as noted above to replace a null path-absolute with "/" or "*".

> *Note: The "no rewrite" rule prevents the proxy from changing the meaning of the request when the origin server is improperly using a non-reserved URI character for a reserved purpose. Implementors need to be aware that some pre-HTTP/1.1 proxies have been known to rewrite the request-target.

HTTP does not place a pre-defined limit on the length of a request-target. A server MUST be prepared to receive URIs of unbounded length and respond with the 414 (URI Too Long) status code if the received request-target would be longer than the server wishes to handle (see Section 8.4.15 of [Part2]).
Various ad-hoc limitations on request-target length are found in practice. It is RECOMMENDED that all HTTP senders and recipients support request-target lengths of 8000 or more octets.

> *Note: Fragments ([RFC3986], Section 3.5) are not part of the request-target and thus will not be transmitted in an HTTP request.

4.2. The Resource Identified by a Request

The exact resource identified by an Internet request is determined by examining both the request-target and the Host header field.
An origin server that does not allow resources to differ by the requested host MAY ignore the Host header field value when determining the resource identified by an HTTP/1.1 request. (But see Appendix Appendix B.1.1 for other requirements on Host support in HTTP/1.1.)
An origin server that does differentiate resources based on the host requested (sometimes referred to as virtual hosts or vanity host names)

MUST use the following rules for determining the requested resource on an HTTP/1.1 request:

1. If request-target is an absolute-URI, the host is part of the request-target. Any Host header field value in the request MUST be ignored.

2. If the request-target is not an absolute-URI, and the request includes a Host header field, the host is determined by the Host header field value.

3. If the host as determined by rule 1 or 2 is not a valid host on the server, the response MUST be a 400 (Bad Request) error message.

Recipients of an HTTP/1.0 request that lacks a Host header field MAY attempt to use heuristics (e.g., examination of the URI path for something unique to a particular host) in order to determine what exact resource is being requested.

4.3. Effective Request URI

HTTP requests often do not carry the absolute URI ([RFC3986], Section 4.3) for the target resource; instead, the URI needs to be inferred from the request-target, Host header field, and connection context. The result of this process is called the "effective request URI". The "target resource" is the resource identified by the effective request URI.
If the request-target is an absolute-URI, then the effective request URI is the request-target.
If the request-target uses the path-absolute form or the asterisk form, and the Host header field is present, then the effective request URI is constructed by concatenating

* the scheme name: "http" if the request was received over an insecure TCP connection, or "https" when received over a SSL/TLS-secured TCP connection,

* the octet sequence "://",

* the authority component, as specified in the Host header field (Section 9.4), and

* the request-target obtained from the Request-Line, unless the request-target is just the asterisk "*".

If the request-target uses the path-absolute form or the asterisk form, and the Host header field is not present, then the effective request URI is undefined.

Otherwise, when request-target uses the authority form, the effective request URI is undefined.
Example 1: the effective request URI for the message

```
GET /pub/WWW/TheProject.html HTTP/1.1
Host: www.example.org:8080
```

(received over an insecure TCP connection) is "http", plus "://", plus the authority component "www.example.org:8080", plus the request-target "/pub/WWW/TheProject.html", thus "http://www.example.org:8080/pub/WWW/TheProject.html".
Example 2: the effective request URI for the message

```
GET * HTTP/1.1
Host: www.example.org
```

(received over an SSL/TLS secured TCP connection) is "https", plus "://", plus the authority component "www.example.org", thus "https://www.example.org".
Effective request URIs are compared using the rules described in Section 2.6.3, except that empty path components MUST NOT be treated as equivalent to an absolute path of "/".

5. Response

After receiving and interpreting a request message, a server responds with an HTTP response message.

```
Response      = Status-Line               ; Section 5.1
                *( header-field CRLF )    ; Section 3.2
                CRLF
                [ message-body ]          ; Section 3.3
```

5.1. Status-Line

The first line of a Response message is the Status-Line, consisting of the protocol version, a space (SP), the status code, another space, a possibly-empty textual phrase describing the status code, and ending with CRLF.

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

5.1.1. Status Code and Reason Phrase

The Status-Code element is a 3-digit integer result code of the attempt to understand and satisfy the request. These codes are fully defined in Section 8 of [Part2]. The Reason Phrase exists for the sole purpose of providing a textual description associated with the numeric status

code, out of deference to earlier Internet application protocols that were more frequently used with interactive text clients. A client SHOULD ignore the content of 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 - Request received, continuing process

*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

```
Status-Code    = 3DIGIT
Reason-Phrase  = *( WSP / VCHAR / obs-text )
```

6. Protocol Parameters

6.1. Date/Time Formats: Full Date

HTTP applications have historically allowed three different formats for date/time stamps. However, the preferred format is a fixed-length subset of that defined by [RFC1123]:

```
Sun, 06 Nov 1994 08:49:37 GMT  ; RFC 1123
```

The other formats are described here only for compatibility with obsolete implementations.

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

HTTP/1.1 clients and servers that parse a date value MUST accept all three formats (for compatibility with HTTP/1.0), though they MUST only generate the RFC 1123 format for representing HTTP-date values in header fields. See Appendix Appendix A for further information.
All HTTP date/time stamps MUST be represented in Greenwich Mean Time (GMT), without exception. For the purposes of HTTP, GMT is exactly equal to UTC (Coordinated Universal Time). This is indicated in the first two formats by the inclusion of "GMT" as the three-letter

abbreviation for time zone, and MUST be assumed when reading the asctime format. HTTP-date is case sensitive and MUST NOT include additional whitespace beyond that specifically included as SP in the grammar.

```
  HTTP-date    = rfc1123-date / obs-date
```

Preferred format:

```
  rfc1123-date = day-name "," SP date1 SP time-of-day SP GMT
  ; fixed length subset of the format defined in
  ; Section 5.2.14 of [RFC1123]

  day-name     = %x4D.6F.6E ; "Mon", case-sensitive
               / %x54.75.65 ; "Tue", case-sensitive
               / %x57.65.64 ; "Wed", case-sensitive
               / %x54.68.75 ; "Thu", case-sensitive
               / %x46.72.69 ; "Fri", case-sensitive
               / %x53.61.74 ; "Sat", case-sensitive
               / %x53.75.6E ; "Sun", case-sensitive

  date1        = day SP month SP year
               ; e.g., 02 Jun 1982

  day          = 2DIGIT
  month        = %x4A.61.6E ; "Jan", case-sensitive
               / %x46.65.62 ; "Feb", case-sensitive
               / %x4D.61.72 ; "Mar", case-sensitive
               / %x41.70.72 ; "Apr", case-sensitive
               / %x4D.61.79 ; "May", case-sensitive
               / %x4A.75.6E ; "Jun", case-sensitive
               / %x4A.75.6C ; "Jul", case-sensitive
               / %x41.75.67 ; "Aug", case-sensitive
               / %x53.65.70 ; "Sep", case-sensitive
               / %x4F.63.74 ; "Oct", case-sensitive
               / %x4E.6F.76 ; "Nov", case-sensitive
               / %x44.65.63 ; "Dec", case-sensitive
  year         = 4DIGIT

  GMT   = %x47.4D.54 ; "GMT", case-sensitive

  time-of-day  = hour ":" minute ":" second
                 ; 00:00:00 - 23:59:59

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

The semantics of day-name *[preferred.date.format]*, day *[preferred.date.format]*, month *[preferred.date.format]*, year

[preferred.date.format], and time-of-day *[preferred.date.format]* are the same as those defined for the RFC 5322 constructs with the corresponding name ([RFC5322], Section 3.3).
Obsolete formats:

```
  obs-date     = rfc850-date / asctime-date

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

  day-name-l   = %x4D.6F.6E.64.61.79 ; "Monday", case-sensitive
         / %x54.75.65.73.64.61.79 ; "Tuesday", case-sensitive
         / %x57.65.64.6E.65.73.64.61.79 ; "Wednesday", case-sensitive
         / %x54.68.75.72.73.64.61.79 ; "Thursday", case-sensitive
         / %x46.72.69.64.61.79 ; "Friday", case-sensitive
         / %x53.61.74.75.72.64.61.79 ; "Saturday", case-sensitive
         / %x53.75.6E.64.61.79 ; "Sunday", case-sensitive

  asctime-date = day-name SP date3 SP time-of-day SP year
  date3        = month SP ( 2DIGIT / ( SP 1DIGIT ))
                   ; month day (e.g., Jun  2)
```

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

*Note: HTTP 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.

6.2. Transfer Codings

Transfer-coding values are used to indicate an encoding transformation that has been, can be, or might need to be applied to a payload body in order to ensure "safe transport" through the network. This differs from a content coding in that the transfer-coding is a property of the message rather than a property of the representation that is being transferred.

```
  transfer-coding         = "chunked" ; Section 6.2.1
                          / "compress" ; Section 6.2.2.1
                          / "deflate" ; Section 6.2.2.2
                          / "gzip" ; Section 6.2.2.3
                          / transfer-extension
  transfer-extension      = token *( OWS ";" OWS transfer-parameter )
```

Parameters are in the form of attribute/value pairs.

```
  transfer-parameter      = attribute BWS "=" BWS value
  attribute               = token
  value                   = word
```

All transfer-coding values are case-insensitive. HTTP/1.1 uses transfer-coding values in the TE header field (Section 9.5) and in the Transfer-Encoding header field (Section 9.7).
Transfer-codings are analogous to the Content-Transfer-Encoding values of MIME, which were designed to enable safe transport of binary data over a 7-bit transport service ([RFC2045], Section 6). However, 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 message body length (Section 3.3), or the desire to encrypt data over a shared transport.
A server that receives a request message with a transfer-coding it does not understand SHOULD respond with 501 (Not Implemented) and then close the connection. A server MUST NOT send transfer-codings to an HTTP/1.0 client.

6.2.1. Chunked Transfer Coding

The chunked encoding modifies the body of a message in order to transfer it as a series of chunks, each with its own size indicator, followed by an OPTIONAL trailer containing header fields. This allows dynamically produced content to be transferred along with the information necessary for the recipient to verify that it has received the full message.

```
  Chunked-Body   = *chunk
                   last-chunk
                   trailer-part
                   CRLF

  chunk          = chunk-size *WSP [ chunk-ext ] CRLF
                   chunk-data CRLF
  chunk-size     = 1*HEXDIG
  last-chunk     = 1*("0") *WSP [ chunk-ext ] CRLF

  chunk-ext      = *( ";" *WSP chunk-ext-name
                      [ "=" chunk-ext-val ] *WSP )
  chunk-ext-name = token
  chunk-ext-val  = token / quoted-str-nf
  chunk-data     = 1*OCTET ; a sequence of chunk-size octets
  trailer-part   = *( header-field CRLF )

  quoted-str-nf  = DQUOTE *( qdtext-nf / quoted-pair ) DQUOTE
                 ; like quoted-string, but disallowing line folding
  qdtext-nf      = WSP / %x21 / %x23-5B / %x5D-7E / obs-text
                 ; WSP / <VCHAR except DQUOTE and "\"> / obs-text
```

The chunk-size field is a string of hex digits indicating the size of the chunk-data in octets. The chunked encoding is ended by any chunk whose size is zero, followed by the trailer, which is terminated by an empty line.
The trailer allows the sender to include additional HTTP header fields at the end of the message. The Trailer header field can be used to indicate which header fields are included in a trailer (see Section 9.6).
A server using chunked transfer-coding in a response MUST NOT use the trailer for any header fields unless at least one of the following is true:

1. the request included a TE header field that indicates "trailers" is acceptable in the transfer-coding of the response, as described in Section 9.5; or,

2. the trailer fields consist entirely of optional metadata, and the recipient could use the message (in a manner acceptable to the server where the field originated) without receiving it. In other words, the server that generated the header (often but not always the origin server) is willing to accept the possibility that the trailer fields might be silently discarded along the path to the client.

This requirement prevents an interoperability failure when the message is being received by an HTTP/1.1 (or later) proxy and forwarded to an HTTP/1.0 recipient. It avoids a situation where compliance with the protocol would have necessitated a possibly infinite buffer on the proxy.
A process for decoding the "chunked" transfer-coding can be represented in pseudo-code as:

```
  length := 0
  read chunk-size, chunk-ext (if any) and CRLF
  while (chunk-size > 0) {
     read chunk-data and CRLF
     append chunk-data to decoded-body
     length := length + chunk-size
     read chunk-size and CRLF
  }
  read header-field
  while (header-field not empty) {
     append header-field to existing header fields
     read header-field
  }
  Content-Length := length
  Remove "chunked" from Transfer-Encoding
```

All HTTP/1.1 applications MUST be able to receive and decode the "chunked" transfer-coding and MUST ignore chunk-ext extensions they do not understand.
Since "chunked" is the only transfer-coding required to be understood by HTTP/1.1 recipients, it plays a crucial role in delimiting messages on a persistent connection. Whenever a transfer-coding is applied to a payload body in a request, the final transfer-coding applied MUST be "chunked". If a transfer-coding is applied to a response payload body, then either the final transfer-coding applied MUST be "chunked" or the message MUST be terminated by closing the connection. When the "chunked" transfer-coding is used, it MUST be the last transfer-coding applied to form the message-body. The "chunked" transfer-coding MUST NOT be applied more than once in a message-body.

6.2.2. Compression Codings

The codings defined below can be used to compress the payload of a message.

> *Note: Use of program names for the identification of encoding formats is not desirable and is discouraged for future encodings. Their use here is representative of historical practice, not good design.
>
> *Note: For compatibility with previous implementations of HTTP, applications SHOULD consider "x-gzip" and "x-compress" to be equivalent to "gzip" and "compress" respectively.

6.2.2.1. Compress Coding

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

6.2.2.2. Deflate Coding

The "deflate" format is defined as the "deflate" compression mechanism (described in [RFC1951]) used inside the "zlib" data format ([RFC1950]).

> *Note: Some incorrect implementations send the "deflate" compressed data without the zlib wrapper.

6.2.2.3. Gzip Coding

The "gzip" format is produced by the file compression program "gzip" (GNU zip), as described in [RFC1952]. This format is a Lempel-Ziv coding (LZ77) with a 32 bit CRC.

6.2.3. Transfer Coding Registry

The HTTP Transfer Coding Registry defines the name space for the transfer coding names.
Registrations MUST include the following fields:

- *Name
- *Description
- *Pointer to specification text

Names of transfer codings MUST NOT overlap with names of content codings (Section 2.2 of [Part3]), unless the encoding transformation is identical (as it is the case for the compression codings defined in Section 6.2.2).
Values to be added to this name space require a specification (see "Specification Required" in Section 4.1 of [RFC5226]), and MUST conform to the purpose of transfer coding defined in this section.
The registry itself is maintained at http://www.iana.org/assignments/http-parameters.

6.3. Product Tokens

Product tokens are used to allow communicating applications to identify themselves by software name and version. Most fields using product tokens also allow sub-products 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. They MUST NOT be used for advertising or other non-essential information. Although any token octet 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).

6.4. Quality Values

Both transfer codings (TE request header field, Section 9.5) and content negotiation (Section 5 of [Part3]) use short "floating point"

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

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

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

7. Connections

7.1. Persistent Connections

7.1.1. Purpose

Prior to persistent connections, a separate TCP connection was established for each request, increasing the load on HTTP servers and causing congestion on the Internet. The use of inline images and other associated data often requires a client to make multiple requests of the same server in a short amount of time. Analysis of these performance problems and results from a prototype implementation are available [Pad1995] [Spe]. Implementation experience and measurements of actual HTTP/1.1 implementations show good results [Nie1997]. Alternatives have also been explored, for example, T/TCP [Tou1998]. Persistent HTTP connections have a number of advantages:

*By opening and closing fewer TCP connections, CPU time is saved in routers and hosts (clients, servers, proxies, gateways, tunnels, or caches), and memory used for TCP protocol control blocks can be saved in hosts.

*HTTP requests and responses can be pipelined on a connection. Pipelining allows a client to make multiple requests without waiting for each response, allowing a single TCP connection to be used much more efficiently, with much lower elapsed time.

*Network congestion is reduced by reducing the number of packets caused by TCP opens, and by allowing TCP sufficient time to determine the congestion state of the network.

*Latency on subsequent requests is reduced since there is no time spent in TCP's connection opening handshake.

*HTTP can evolve more gracefully, since errors can be reported without the penalty of closing the TCP connection. Clients using

future versions of HTTP might optimistically try a new feature, but if communicating with an older server, retry with old semantics after an error is reported.

HTTP implementations SHOULD implement persistent connections.

7.1.2. Overall Operation

A significant difference between HTTP/1.1 and earlier versions of HTTP is that persistent connections are the default behavior of any HTTP connection. That is, unless otherwise indicated, the client SHOULD assume that the server will maintain a persistent connection, even after error responses from the server.
Persistent connections provide a mechanism by which a client and a server can signal the close of a TCP connection. This signaling takes place using the Connection header field (Section 9.1). Once a close has been signaled, the client MUST NOT send any more requests on that connection.

7.1.2.1. Negotiation

An HTTP/1.1 server MAY assume that a HTTP/1.1 client intends to maintain a persistent connection unless a Connection header field including the connection-token "close" was sent in the request. If the server chooses to close the connection immediately after sending the response, it SHOULD send a Connection header field including the connection-token "close".
An HTTP/1.1 client MAY expect a connection to remain open, but would decide to keep it open based on whether the response from a server contains a Connection header field with the connection-token close. In case the client does not want to maintain a connection for more than that request, it SHOULD send a Connection header field including the connection-token close.
If either the client or the server sends the close token in the Connection header field, that request becomes the last one for the connection.
Clients and servers SHOULD NOT assume that a persistent connection is maintained for HTTP versions less than 1.1 unless it is explicitly signaled. See Appendix Appendix B.1.2 for more information on backward compatibility with HTTP/1.0 clients.
In order to remain persistent, all messages on the connection MUST have a self-defined message length (i.e., one not defined by closure of the connection), as described in Section 3.3.

7.1.2.2. Pipelining

A client that supports persistent connections MAY "pipeline" its requests (i.e., send multiple requests without waiting for each

response). A server MUST send its responses to those requests in the same order that the requests were received.
Clients which assume persistent connections and pipeline immediately after connection establishment SHOULD be prepared to retry their connection if the first pipelined attempt fails. If a client does such a retry, it MUST NOT pipeline before it knows the connection is persistent. Clients MUST also be prepared to resend their requests if the server closes the connection before sending all of the corresponding responses.
Clients SHOULD NOT pipeline requests using non-idempotent request methods or non-idempotent sequences of request methods (see Section 7.1.2 of [Part2]). Otherwise, a premature termination of the transport connection could lead to indeterminate results. A client wishing to send a non-idempotent request SHOULD wait to send that request until it has received the response status line for the previous request.

7.1.3. Proxy Servers

It is especially important that proxies correctly implement the properties of the Connection header field as specified in Section 9.1.
The proxy server MUST signal persistent connections separately with its clients and the origin servers (or other proxy servers) that it connects to. Each persistent connection applies to only one transport link.
A proxy server MUST NOT establish a HTTP/1.1 persistent connection with an HTTP/1.0 client (but see Section 19.7.1 of [RFC2068] for information and discussion of the problems with the Keep-Alive header field implemented by many HTTP/1.0 clients).

7.1.3.1. End-to-end and Hop-by-hop Header Fields

For the purpose of defining the behavior of caches and non-caching proxies, we divide HTTP header fields into two categories:

- End-to-end header fields, which are transmitted to the ultimate recipient of a request or response. End-to-end header fields in responses MUST be stored as part of a cache entry and MUST be transmitted in any response formed from a cache entry.

- Hop-by-hop header fields, which are meaningful only for a single transport-level connection, and are not stored by caches or forwarded by proxies.

The following HTTP/1.1 header fields are hop-by-hop header fields:

- Connection

- Keep-Alive

- Proxy-Authenticate

*Proxy-Authorization

*TE

*Trailer

*Transfer-Encoding

*Upgrade

All other header fields defined by HTTP/1.1 are end-to-end header fields.
Other hop-by-hop header fields MUST be listed in a Connection header field (Section 9.1).

7.1.3.2. Non-modifiable Header Fields

Some features of HTTP/1.1, such as Digest Authentication, depend on the value of certain end-to-end header fields. A non-transforming proxy SHOULD NOT modify an end-to-end header field unless the definition of that header field requires or specifically allows that.
A non-transforming proxy MUST NOT modify any of the following fields in a request or response, and it MUST NOT add any of these fields if not already present:

*Content-Location

*Content-MD5

*ETag

*Last-Modified

A non-transforming proxy MUST NOT modify any of the following fields in a response:

*Expires

but it MAY add any of these fields if not already present. If an Expires header field is added, it MUST be given a field-value identical to that of the Date header field in that response.
A proxy MUST NOT modify or add any of the following fields in a message that contains the no-transform cache-control directive, or in any request:

*Content-Encoding

*Content-Range

*Content-Type

A transforming proxy MAY modify or add these fields to a message that does not include no-transform, but if it does so, it MUST add a Warning 214 (Transformation applied) if one does not already appear in the message (see Section 3.6 of [Part6]).

> *Warning: Unnecessary modification of end-to-end header fields might cause authentication failures if stronger authentication mechanisms are introduced in later versions of HTTP. Such authentication mechanisms MAY rely on the values of header fields not listed here.

A non-transforming proxy MUST preserve the message payload ([Part3]), though it MAY change the message-body through application or removal of a transfer-coding (Section 6.2).

7.1.4. Practical Considerations

Servers will usually have some time-out value beyond which they will no longer maintain an inactive connection. Proxy servers might make this a higher value since it is likely that the client will be making more connections through the same server. The use of persistent connections places no requirements on the length (or existence) of this time-out for either the client or the server.

When a client or server wishes to time-out it SHOULD issue a graceful close on the transport connection. Clients and servers SHOULD both constantly watch for the other side of the transport close, and respond to it as appropriate. If a client or server does not detect the other side's close promptly it could cause unnecessary resource drain on the network.

A client, server, or proxy MAY close the transport connection at any time. For example, a client might have started to send a new request at the same time that the server has decided to close the "idle" connection. From the server's point of view, the connection is being closed while it was idle, but from the client's point of view, a request is in progress.

This means that clients, servers, and proxies MUST be able to recover from asynchronous close events. Client software SHOULD reopen the transport connection and retransmit the aborted sequence of requests without user interaction so long as the request sequence is idempotent (see Section 7.1.2 of [Part2]). Non-idempotent request methods or sequences MUST NOT be automatically retried, although user agents MAY offer a human operator the choice of retrying the request(s). Confirmation by user-agent software with semantic understanding of the application MAY substitute for user confirmation. The automatic retry SHOULD NOT be repeated if the second sequence of requests fails.

Servers SHOULD always respond to at least one request per connection, if at all possible. Servers SHOULD NOT close a connection in the middle of transmitting a response, unless a network or client failure is suspected.

Clients (including proxies) SHOULD limit the number of simultaneous connections that they maintain to a given server (including proxies). Previous revisions of HTTP gave a specific number of connections as a ceiling, but this was found to be impractical for many applications. As a result, this specification does not mandate a particular maximum number of connections, but instead encourages clients to be conservative when opening multiple connections.
In particular, while using multiple connections avoids the "head-of-line blocking" problem (whereby a request that takes significant server-side processing and/or has a large payload can block subsequent requests on the same connection), each connection used consumes server resources (sometimes significantly), and furthermore using multiple connections can cause undesirable side effects in congested networks. Note that servers might reject traffic that they deem abusive, including an excessive number of connections from a client.

7.2. Message Transmission Requirements

7.2.1. Persistent Connections and Flow Control

HTTP/1.1 servers SHOULD maintain persistent connections and use TCP's flow control mechanisms to resolve temporary overloads, rather than terminating connections with the expectation that clients will retry. The latter technique can exacerbate network congestion.

7.2.2. Monitoring Connections for Error Status Messages

An HTTP/1.1 (or later) client sending a message-body SHOULD monitor the network connection for an error status code while it is transmitting the request. If the client sees an error status code, it SHOULD immediately cease transmitting the body. If the body is being sent using a "chunked" encoding (Section 6.2), a zero length chunk and empty trailer MAY be used to prematurely mark the end of the message. If the body was preceded by a Content-Length header field, the client MUST close the connection.

7.2.3. Use of the 100 (Continue) Status

The purpose of the 100 (Continue) status code (see Section 8.1.1 of [Part2]) is to allow a client that is sending a request message with a request body to determine if the origin server is willing to accept the request (based on the request header fields) before the client sends the request body. In some cases, it might either be inappropriate or highly inefficient for the client to send the body if the server will reject the message without looking at the body.
Requirements for HTTP/1.1 clients:

* If a client will wait for a 100 (Continue) response before sending the request body, it MUST send an Expect header field (Section 9.2 of [Part2]) with the "100-continue" expectation.

* A client MUST NOT send an Expect header field (Section 9.2 of [Part2]) with the "100-continue" expectation if it does not intend to send a request body.

Because of the presence of older implementations, the protocol allows ambiguous situations in which a client might send "Expect: 100-continue" without receiving either a 417 (Expectation Failed) or a 100 (Continue) status code. Therefore, when a client sends this header field to an origin server (possibly via a proxy) from which it has never seen a 100 (Continue) status code, the client SHOULD NOT wait for an indefinite period before sending the request body.

Requirements for HTTP/1.1 origin servers:

* Upon receiving a request which includes an Expect header field with the "100-continue" expectation, an origin server MUST either respond with 100 (Continue) status code and continue to read from the input stream, or respond with a final status code. The origin server MUST NOT wait for the request body before sending the 100 (Continue) response. If it responds with a final status code, it MAY close the transport connection or it MAY continue to read and discard the rest of the request. It MUST NOT perform the request method if it returns a final status code.

* An origin server SHOULD NOT send a 100 (Continue) response if the request message does not include an Expect header field with the "100-continue" expectation, and MUST NOT send a 100 (Continue) response if such a request comes from an HTTP/1.0 (or earlier) client. There is an exception to this rule: for compatibility with [RFC2068], a server MAY send a 100 (Continue) status code in response to an HTTP/1.1 PUT or POST request that does not include an Expect header field with the "100-continue" expectation. This exception, the purpose of which is to minimize any client processing delays associated with an undeclared wait for 100 (Continue) status code, applies only to HTTP/1.1 requests, and not to requests with any other HTTP-version value.

* An origin server MAY omit a 100 (Continue) response if it has already received some or all of the request body for the corresponding request.

* An origin server that sends a 100 (Continue) response MUST ultimately send a final status code, once the request body is received and processed, unless it terminates the transport connection prematurely.

* If an origin server receives a request that does not include an Expect header field with the "100-continue" expectation, the request includes a request body, and the server responds with a final status code before reading the entire request body from the

transport connection, then the server SHOULD NOT close the transport connection until it has read the entire request, or until the client closes the connection. Otherwise, the client might not reliably receive the response message. However, this requirement is not be construed as preventing a server from defending itself against denial-of-service attacks, or from badly broken client implementations.

Requirements for HTTP/1.1 proxies:

*If a proxy receives a request that includes an Expect header field with the "100-continue" expectation, and the proxy either knows that the next-hop server complies with HTTP/1.1 or higher, or does not know the HTTP version of the next-hop server, it MUST forward the request, including the Expect header field.

*If the proxy knows that the version of the next-hop server is HTTP/1.0 or lower, it MUST NOT forward the request, and it MUST respond with a 417 (Expectation Failed) status code.

*Proxies SHOULD maintain a cache recording the HTTP version numbers received from recently-referenced next-hop servers.

*A proxy MUST NOT forward a 100 (Continue) response if the request message was received from an HTTP/1.0 (or earlier) client and did not include an Expect header field with the "100-continue" expectation. This requirement overrides the general rule for forwarding of 1xx responses (see Section 8.1 of [Part2]).

7.2.4. Client Behavior if Server Prematurely Closes Connection

If an HTTP/1.1 client sends a request which includes a request body, but which does not include an Expect header field with the "100-continue" expectation, and if the client is not directly connected to an HTTP/1.1 origin server, and if the client sees the connection close before receiving a status line from the server, the client SHOULD retry the request. If the client does retry this request, it MAY use the following "binary exponential backoff" algorithm to be assured of obtaining a reliable response:

1. Initiate a new connection to the server

2. Transmit the request-line, header fields, and the CRLF that indicates the end of header fields.

3. Initialize a variable R to the estimated round-trip time to the server (e.g., based on the time it took to establish the connection), or to a constant value of 5 seconds if the round-trip time is not available.

4. Compute T = R * (2**N), where N is the number of previous retries of this request.

5. Wait either for an error response from the server, or for T seconds (whichever comes first)

6. If no error response is received, after T seconds transmit the body of the request.

7. If client sees that the connection is closed prematurely, repeat from step 1 until the request is accepted, an error response is received, or the user becomes impatient and terminates the retry process.

If at any point an error status code is received, the client

*SHOULD NOT continue and

*SHOULD close the connection if it has not completed sending the request message.

8. Miscellaneous notes that might disappear

8.1. Scheme aliases considered harmful

8.2. Use of HTTP for proxy communication

8.3. Interception of HTTP for access control

8.4. Use of HTTP by other protocols

8.5. Use of HTTP by media type specification

9. Header Field Definitions

This section defines the syntax and semantics of HTTP header fields related to message framing and transport protocols.

9.1. Connection

The "Connection" header field allows the sender to specify options that are desired only for that particular connection. Such connection options MUST be removed or replaced before the message can be forwarded downstream by a proxy or gateway. This mechanism also allows the sender to indicate which HTTP header fields used in the message are only intended for the immediate recipient ("hop-by-hop"), as opposed to all recipients on the chain ("end-to-end"), enabling the message to be self-descriptive and allowing future connection-specific extensions to be deployed in HTTP without fear that they will be blindly forwarded by previously deployed intermediaries.

The Connection header field's value has the following grammar:

```
  Connection       = "Connection" ":" OWS Connection-v
  Connection-v     = 1#connection-token
  connection-token = token
```

A proxy or gateway MUST parse a received Connection header field before a message is forwarded and, for each connection-token in this field, remove any header field(s) from the message with the same name as the connection-token, and then remove the Connection header field itself or replace it with the sender's own connection options for the forwarded message.

A sender MUST NOT include field-names in the Connection header field-value for fields that are defined as expressing constraints for all recipients in the request or response chain, such as the Cache-Control header field (Section 3.2 of [Part6]).

The connection options do not have to correspond to a header field present in the message, since a connection-specific header field might not be needed if there are no parameters associated with that connection option. Recipients that trigger certain connection behavior based on the presence of connection options MUST do so based on the presence of the connection-token rather than only the presence of the optional header field. In other words, if the connection option is received as a header field but not indicated within the Connection field-value, then the recipient MUST ignore the connection-specific header field because it has likely been forwarded by an intermediary that is only partially compliant.

When defining new connection options, specifications ought to carefully consider existing deployed header fields and ensure that the new connection-token does not share the same name as an unrelated header field that might already be deployed. Defining a new connection-token essentially reserves that potential field-name for carrying additional information related to the connection option, since it would be unwise for senders to use that field-name for anything else.

HTTP/1.1 defines the "close" connection option for the sender to signal that the connection will be closed after completion of the response. For example,

```
  Connection: close
```

in either the request or the response header fields indicates that the connection SHOULD NOT be considered "persistent" (Section 7.1) after the current request/response is complete.

An HTTP/1.1 client that does not support persistent connections MUST include the "close" connection option in every request message.

An HTTP/1.1 server that does not support persistent connections MUST include the "close" connection option in every response message that does not have a 1xx (Informational) status code.

9.2. Content-Length

The "Content-Length" header field indicates the size of the message-body, in decimal number of octets, for any message other than a response to a HEAD request or a response with a status code of 304. In the case of a response to a HEAD request, Content-Length indicates the size of the payload body (not including any potential transfer-coding) that would have been sent had the request been a GET. In the case of a 304 (Not Modified) response to a GET request, Content-Length indicates the size of the payload body (not including any potential transfer-coding) that would have been sent in a 200 (OK) response.

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

An example is

```
  Content-Length: 3495
```

Implementations SHOULD use this field to indicate the message-body length when no transfer-coding is being applied and the payload's body length can be determined prior to being transferred. Section 3.3 describes how recipients determine the length of a message-body. Any Content-Length greater than or equal to zero is a valid value. Note that the use of this field in HTTP is significantly different from the corresponding definition in MIME, where it is an optional field used within the "message/external-body" content-type.

9.3. Date

The "Date" header field represents the date and time at which the message was originated, having the same semantics as the Origination Date Field (orig-date) defined in Section 3.6.1 of [RFC5322]. The field value is an HTTP-date, as described in Section 6.1; it MUST be sent in rfc1123-date format.

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

An example is

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

Origin servers MUST include a Date header field in all responses, except in these cases:

1. If the response status code is 100 (Continue) or 101 (Switching Protocols), the response MAY include a Date header field, at the server's option.

2. If the response status code conveys a server error, e.g., 500 (Internal Server Error) or 503 (Service Unavailable), and it is inconvenient or impossible to generate a valid Date.

3. If the server does not have a clock that can provide a reasonable approximation of the current time, its responses MUST NOT include a Date header field. In this case, the rules in Section 9.3.1 MUST be followed.

A received message that does not have a Date header field MUST be assigned one by the recipient if the message will be cached by that recipient.
Clients can use the Date header field as well; in order to keep request messages small, they are advised not to include it when it doesn't convey any useful information (as it is usually the case for requests that do not contain a payload).
The HTTP-date sent in a Date header field SHOULD NOT represent a date and time subsequent to the generation of the message. It SHOULD represent the best available approximation of the date and time of message generation, unless the implementation has no means of generating a reasonably accurate date and time. In theory, the date ought to represent the moment just before the payload is generated. In practice, the date can be generated at any time during the message origination without affecting its semantic value.

9.3.1. Clockless Origin Server Operation

Some origin server implementations might not have a clock available. An origin server without a clock MUST NOT assign Expires or Last-Modified values to a response, unless these values were associated with the resource by a system or user with a reliable clock. It MAY assign an Expires value that is known, at or before server configuration time, to be in the past (this allows "pre-expiration" of responses without storing separate Expires values for each resource).

9.4. Host

The "Host" header field in a request provides the host and port information from the target resource's URI, enabling the origin server to distinguish between resources while servicing requests for multiple host names on a single IP address. Since the Host field-value is critical information for handling a request, it SHOULD be sent as the first header field following the Request-Line.

```
Host   = "Host" ":" OWS Host-v
Host-v = uri-host [ ":" port ] ; Section 2.6.1
```

A client MUST send a Host header field in all HTTP/1.1 request messages. If the target resource's URI includes an authority component, then the Host field-value MUST be identical to that authority component

after excluding any userinfo (Section 2.6.1). If the authority component is missing or undefined for the target resource's URI, then the Host header field MUST be sent with an empty field-value.
For example, a GET request to the origin server for <http://www.example.org/pub/WWW/> would begin with:

```
GET /pub/WWW/ HTTP/1.1
Host: www.example.org
```

The Host header field MUST be sent in an HTTP/1.1 request even if the request-target is in the form of an absolute-URI, since this allows the Host information to be forwarded through ancient HTTP/1.0 proxies that might not have implemented Host.
When an HTTP/1.1 proxy receives a request with a request-target in the form of an absolute-URI, the proxy MUST ignore the received Host header field (if any) and instead replace it with the host information of the request-target. When a proxy forwards a request, it MUST generate the Host header field based on the received absolute-URI rather than the received Host.
Since the Host header field acts as an application-level routing mechanism, it is a frequent target for malware seeking to poison a shared cache or redirect a request to an unintended server. An interception proxy is particularly vulnerable if it relies on the Host header field value for redirecting requests to internal servers, or for use as a cache key in a shared cache, without first verifying that the intercepted connection is targeting a valid IP address for that host.
A server MUST respond with a 400 (Bad Request) status code to any HTTP/1.1 request message that lacks a Host header field and to any request message that contains more than one Host header field or a Host header field with an invalid field-value.
See Sections 4.2 and Appendix B.1.1 for other requirements relating to Host.

9.5. TE

The "TE" header field indicates what extension transfer-codings it is willing to accept in the response, and whether or not it is willing to accept trailer fields in a chunked transfer-coding.
Its value consists of the keyword "trailers" and/or a comma-separated list of extension transfer-coding names with optional accept parameters (as described in Section 6.2).

```
TE        = "TE" ":" OWS TE-v
TE-v      = #t-codings
t-codings = "trailers" / ( transfer-extension [ te-params ] )
te-params = OWS ";" OWS "q=" qvalue *( te-ext )
te-ext    = OWS ";" OWS token [ "=" word ]
```

The presence of the keyword "trailers" indicates that the client is willing to accept trailer fields in a chunked transfer-coding, as defined in Section 6.2.1. This keyword is reserved for use with transfer-coding values even though it does not itself represent a transfer-coding.
Examples of its use are:

```
TE: deflate
TE:
TE: trailers, deflate;q=0.5
```

The TE header field only applies to the immediate connection. Therefore, the keyword MUST be supplied within a Connection header field (Section 9.1) whenever TE is present in an HTTP/1.1 message.
A server tests whether a transfer-coding is acceptable, according to a TE field, using these rules:

1. The "chunked" transfer-coding is always acceptable. If the keyword "trailers" is listed, the client indicates that it is willing to accept trailer fields in the chunked response on behalf of itself and any downstream clients. The implication is that, if given, the client is stating that either all downstream clients are willing to accept trailer fields in the forwarded response, or that it will attempt to buffer the response on behalf of downstream recipients.

   Note: HTTP/1.1 does not define any means to limit the size of a chunked response such that a client can be assured of buffering the entire response.

2. If the transfer-coding being tested is one of the transfer-codings listed in the TE field, then it is acceptable unless it is accompanied by a qvalue of 0. (As defined in Section 6.4, a qvalue of 0 means "not acceptable".)

3. If multiple transfer-codings are acceptable, then the acceptable transfer-coding with the highest non-zero qvalue is preferred. The "chunked" transfer-coding always has a qvalue of 1.

If the TE field-value is empty or if no TE field is present, the only transfer-coding is "chunked". A message with no transfer-coding is always acceptable.

9.6. Trailer

The "Trailer" header field indicates that the given set of header fields is present in the trailer of a message encoded with chunked transfer-coding.

```
  Trailer   = "Trailer" ":" OWS Trailer-v
  Trailer-v = 1#field-name
```

An HTTP/1.1 message SHOULD include a Trailer header field in a message using chunked transfer-coding with a non-empty trailer. Doing so allows the recipient to know which header fields to expect in the trailer.
If no Trailer header field is present, the trailer SHOULD NOT include any header fields. See Section 6.2.1 for restrictions on the use of trailer fields in a "chunked" transfer-coding.
Message header fields listed in the Trailer header field MUST NOT include the following header fields:

- *Transfer-Encoding
- *Content-Length
- *Trailer

9.7. Transfer-Encoding

The "Transfer-Encoding" header field indicates what transfer-codings (if any) have been applied to the message body. It differs from Content-Encoding (Section 2.2 of [Part3]) in that transfer-codings are a property of the message (and therefore are removed by intermediaries), whereas content-codings are not.

```
  Transfer-Encoding   = "Transfer-Encoding" ":" OWS
                        Transfer-Encoding-v
  Transfer-Encoding-v = 1#transfer-coding
```

Transfer-codings are defined in Section 6.2. An example is:

```
  Transfer-Encoding: chunked
```

If multiple encodings have been applied to a representation, the transfer-codings MUST be listed in the order in which they were applied. Additional information about the encoding parameters MAY be provided by other header fields not defined by this specification.
Many older HTTP/1.0 applications do not understand the Transfer-Encoding header field.

9.8. Upgrade

The "Upgrade" header field allows the client to specify what additional communication protocols it would like to use, if the server chooses to switch protocols. Servers can use it to indicate what protocols they are willing to switch to.

```
  Upgrade   = "Upgrade" ":" OWS Upgrade-v
  Upgrade-v = 1#product
```

For example,

```
  Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11
```

The Upgrade header field is intended to provide a simple mechanism for transition from HTTP/1.1 to some other, incompatible protocol. It does so by allowing the client to advertise its desire to use another protocol, such as a later version of HTTP with a higher major version number, even though the current request has been made using HTTP/1.1. This eases the difficult transition between incompatible protocols by allowing the client to initiate a request in the more commonly supported protocol while indicating to the server that it would like to use a "better" protocol if available (where "better" is determined by the server, possibly according to the nature of the request method or target resource).

The Upgrade header field only applies to switching application-layer protocols upon the existing transport-layer connection. Upgrade cannot be used to insist on a protocol change; its acceptance and use by the server is optional. The capabilities and nature of the application-layer communication after the protocol change is entirely dependent upon the new protocol chosen, although the first action after changing the protocol MUST be a response to the initial HTTP request containing the Upgrade header field.

The Upgrade header field only applies to the immediate connection. Therefore, the upgrade keyword MUST be supplied within a Connection header field (Section 9.1) whenever Upgrade is present in an HTTP/1.1 message.

The Upgrade header field cannot be used to indicate a switch to a protocol on a different connection. For that purpose, it is more appropriate to use a 3xx redirection response (Section 8.3 of [Part2]).

Servers MUST include the "Upgrade" header field in 101 (Switching Protocols) responses to indicate which protocol(s) are being switched to, and MUST include it in 426 (Upgrade Required) responses to indicate acceptable protocols to upgrade to. Servers MAY include it in any other response to indicate that they are willing to upgrade to one of the specified protocols.

This specification only defines the protocol name "HTTP" for use by the family of Hypertext Transfer Protocols, as defined by the HTTP version rules of Section 2.5 and future updates to this specification. Additional tokens can be registered with IANA using the registration procedure defined below.

9.8.1. Upgrade Token Registry

The HTTP Upgrade Token Registry defines the name space for product tokens used to identify protocols in the Upgrade header field. Each registered token is associated with contact information and an optional set of specifications that details how the connection will be processed after it has been upgraded.

Registrations are allowed on a First Come First Served basis as described in Section 4.1 of [RFC5226]. The specifications need not be IETF documents or be subject to IESG review. Registrations are subject to the following rules:

1. A token, once registered, stays registered forever.

2. The registration MUST name a responsible party for the registration.

3. The registration MUST name a point of contact.

4. The registration MAY name a set of specifications associated with that token. Such specifications need not be publicly available.

5. The responsible party MAY change the registration at any time. The IANA will keep a record of all such changes, and make them available upon request.

6. The responsible party for the first registration of a "product" token MUST approve later registrations of a "version" token together with that "product" token before they can be registered.

7. If absolutely required, the IESG MAY reassign the responsibility for a token. This will normally only be used in the case when a responsible party cannot be contacted.

9.9. Via

The "Via" header field MUST be sent by a proxy or gateway to indicate the intermediate protocols and recipients 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 used by email systems (Section 3.6.7 of [RFC5322]) and is intended to be used for tracking message forwards, avoiding request loops, and identifying the protocol capabilities of all senders along the request/response chain.

```
  Via               = "Via" ":" OWS Via-v
  Via-v             = 1#( received-protocol RWS received-by
                          [ RWS comment ] )
  received-protocol = [ protocol-name "/" ] protocol-version
  protocol-name     = token
  protocol-version  = token
  received-by       = ( uri-host [ ":" port ] ) / pseudonym
  pseudonym         = token
```

The received-protocol indicates the protocol version of the message received by the server or client along each segment of the request/

response chain. The received-protocol version is appended to the Via field value when the message is forwarded so that information about the protocol capabilities of upstream applications remains visible to all recipients.
The protocol-name is excluded if and only if it would be "HTTP". The received-by field is normally the host and optional port number of a recipient server or client that subsequently forwarded the message. However, if the real host is considered to be sensitive information, it MAY be replaced by a pseudonym. If the port is not given, it MAY be assumed to be the default port of the received-protocol.
Multiple Via field values represent each proxy or gateway that has forwarded the message. Each recipient MUST append its information such that the end result is ordered according to the sequence of forwarding applications.
Comments MAY be used in the Via header field to identify the software of each recipient, analogous to the User-Agent and Server header fields. However, all comments in the Via field are optional and MAY be removed by any recipient prior to forwarding the message.
For example, a request message could be sent from an HTTP/1.0 user agent to an internal proxy code-named "fred", which uses HTTP/1.1 to forward the request to a public proxy at p.example.net, which completes the request by forwarding it to the origin server at www.example.com. The request received by www.example.com would then have the following Via header field:

```
  Via: 1.0 fred, 1.1 p.example.net (Apache/1.1)
```

A proxy or gateway used as a portal through a network firewall SHOULD NOT forward the names and ports of hosts within the firewall region unless it is explicitly enabled to do so. If not enabled, the received-by host of any host behind the firewall SHOULD be replaced by an appropriate pseudonym for that host.
For organizations that have strong privacy requirements for hiding internal structures, a proxy or gateway MAY combine an ordered subsequence of Via header field entries with identical received-protocol values into a single such entry. For example,

```
  Via: 1.0 ricky, 1.1 ethel, 1.1 fred, 1.0 lucy
```

could be collapsed to

```
  Via: 1.0 ricky, 1.1 mertz, 1.0 lucy
```

Senders SHOULD NOT combine multiple entries unless they are all under the same organizational control and the hosts have already been replaced by pseudonyms. Senders MUST NOT combine entries which have different received-protocol values.

10. IANA Considerations

10.1. Header Field Registration

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

Header Field Name	Protocol	Status	Reference
Connection	http	standard	Section 9.1
Content-Length	http	standard	Section 9.2
Date	http	standard	Section 9.3
Host	http	standard	Section 9.4
TE	http	standard	Section 9.5
Trailer	http	standard	Section 9.6
Transfer-Encoding	http	standard	Section 9.7
Upgrade	http	standard	Section 9.8
Via	http	standard	Section 9.9

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

10.2. URI Scheme Registration

The entries for the "http" and "https" URI Schemes in the registry located at http://www.iana.org/assignments/uri-schemes.html shall be updated to point to Sections 2.6.1 and 2.6.2 of this document (see [RFC4395]).

10.3. Internet Media Type Registrations

This document serves as the specification for the Internet media types "message/http" and "application/http". The following is to be registered with IANA (see [RFC4288]).

10.3.1. Internet Media Type message/http

The message/http type can be used to enclose a single HTTP request or response message, provided that it obeys the MIME restrictions for all "message" types regarding line length and encodings.

Type name: message

Subtype name: http

Required parameters: none

Optional parameters:
version, msgtype

version: The HTTP-Version number of the enclosed message (e.g., "1.1"). 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

Interoperability considerations: none

Published specification: This specification (see Section 10.3.1).

Applications that use this media type:
Additional information:
Magic number(s): none

File extension(s): none

Macintosh file type code(s): none

Person and email address to contact for further information: See Authors Section.

Intended usage: COMMON

Restrictions on usage: none

Author/Change controller: IESG

10.3.2. Internet Media Type application/http

The application/http type can be used to enclose a pipeline of one or more HTTP request or response messages (not intermixed).

Type name: application

Subtype name: http

Required parameters: none

Optional parameters: version, msgtype

version:

The HTTP-Version number of the enclosed messages (e.g., "1.1"). 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: HTTP messages enclosed by this type are in "binary" format; use of an appropriate Content-Transfer-Encoding is required when transmitted via E-mail.

Security considerations: none

Interoperability considerations: none

Published specification: This specification (see Section 10.3.2).

Applications that use this media type:
Additional information:
Magic number(s): none

File extension(s): none

Macintosh file type code(s): none

Person and email address to contact for further information: See Authors Section.

Intended usage: COMMON

Restrictions on usage: none

Author/Change controller: IESG

10.4. Transfer Coding Registry

The registration procedure for HTTP Transfer Codings is now defined by Section 6.2.3 of this document.
The HTTP Transfer Codings Registry located at http://www.iana.org/assignments/http-parameters shall be updated with the registrations below:

Name	Description	Reference
chunked	Transfer in a series of chunks	Section 6.2.1
compress	UNIX "compress" program method	Section 6.2.2.1
deflate		

Name	Description	Reference
	"deflate" compression mechanism ([RFC1951]) used inside the "zlib" data format ([RFC1950])	Section 6.2.2.2
gzip	Same as GNU zip [RFC1952]	Section 6.2.2.3

10.5. Upgrade Token Registration

The registration procedure for HTTP Upgrade Tokens — previously defined in Section 7.2 of [RFC2817] — is now defined by Section 9.8.1 of this document.
The HTTP Status Code Registry located at http://www.iana.org/assignments/http-upgrade-tokens/ shall be updated with the registration below:

Value	Description	Reference
HTTP	Hypertext Transfer Protocol	Section 2.5 of this specification

11. Security Considerations

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

11.1. Personal Information

HTTP clients are often privy to large amounts of personal information (e.g., the user's name, location, mail address, passwords, encryption keys, etc.), and SHOULD be very careful to prevent unintentional leakage of this information. We very strongly recommend that a convenient interface be provided for the user to control dissemination of such information, and that designers and implementors be particularly careful in this area. History shows that errors in this area often create serious security and/or privacy problems and generate highly adverse publicity for the implementor's company.

11.2. Abuse of Server Log Information

A server is in the position to save personal data about a user's requests which might identify their reading patterns or subjects of interest. This information is clearly confidential in nature and its handling can be constrained by law in certain countries. People using HTTP 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.

11.3. Attacks Based On File and Path Names

Implementations of HTTP origin servers SHOULD be careful to restrict the documents returned by HTTP requests to be only those that were intended by the server administrators. If an HTTP server translates HTTP URIs directly into file system calls, the server MUST take special care not to serve files that were not intended to be delivered to HTTP clients. For example, UNIX, Microsoft Windows, and other operating systems use ".." as a path component to indicate a directory level above the current one. On such a system, an HTTP server MUST disallow any such construct in the request-target if it would otherwise allow access to a resource outside those intended to be accessible via the HTTP server. Similarly, files intended for reference only internally to the server (such as access control files, configuration files, and script code) MUST be protected from inappropriate retrieval, since they might contain sensitive information. Experience has shown that minor bugs in such HTTP server implementations have turned into security risks.

11.4. DNS Spoofing

Clients using HTTP rely heavily on the Domain Name Service, and are thus generally prone to security attacks based on the deliberate mis-association of IP addresses and DNS names. Clients need to be cautious in assuming the continuing validity of an IP number/DNS name association.

In particular, HTTP clients SHOULD rely on their name resolver for confirmation of an IP number/DNS name association, rather than caching the result of previous host name lookups. Many platforms already can cache host name lookups locally when appropriate, and they SHOULD be configured to do so. It is proper for these lookups to be cached, however, only when the TTL (Time To Live) information reported by the name server makes it likely that the cached information will remain useful.

If HTTP clients cache the results of host name lookups in order to achieve a performance improvement, they MUST observe the TTL information reported by DNS.

If HTTP clients do not observe this rule, they could be spoofed when a previously-accessed server's IP address changes. As network renumbering is expected to become increasingly common [RFC1900], the possibility of this form of attack will grow. Observing this requirement thus reduces this potential security vulnerability.

This requirement also improves the load-balancing behavior of clients for replicated servers using the same DNS name and reduces the likelihood of a user's experiencing failure in accessing sites which use that strategy.

11.5. Proxies and Caching

By their very nature, HTTP proxies are men-in-the-middle, and represent an opportunity for man-in-the-middle attacks. Compromise of the systems on which the proxies run can result in serious security and privacy problems. Proxies have access to security-related information, personal information about individual users and organizations, and proprietary information belonging to users and content providers. A compromised proxy, or a proxy implemented or configured without regard to security and privacy considerations, might be used in the commission of a wide range of potential attacks.

Proxy operators need to protect the systems on which proxies run as they would protect any system that contains or transports sensitive information. In particular, log information gathered at proxies often contains highly sensitive personal information, and/or information about organizations. Log information needs to be carefully guarded, and appropriate guidelines for use need to be developed and followed. (Section 11.2).

Proxy implementors need to consider the privacy and security implications of their design and coding decisions, and of the configuration options they provide to proxy operators (especially the default configuration).

Users of a proxy need to be aware that proxies are no trustworthier than the people who run them; HTTP itself cannot solve this problem. The judicious use of cryptography, when appropriate, might suffice to protect against a broad range of security and privacy attacks. Such cryptography is beyond the scope of the HTTP/1.1 specification.

11.6. Denial of Service Attacks on Proxies

They exist. They are hard to defend against. Research continues. Beware.

12. Acknowledgments

HTTP has evolved considerably over the 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, John Franks, Jean-Francois Groff, Phillip M. Hallam-Baker, Hakon W. Lie, Ari Luotonen, Rob McCool, Lou Montulli, Dave Raggett, Tony Sanders, and Marc VanHeyningen deserve special recognition for their efforts in defining early aspects of the protocol.

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, Josh Cohen, Mike Cowlishaw, Roman Czyborra, Michael A. Dolan, Daniel DuBois, David J. Fiander, Alan Freier, Marc Hedlund, Greg Herlihy, Koen Holtman, Alex Hopmann, Bob Jernigan, Shel Kaphan, Rohit Khare, John Klensin, Martijn Koster, Alexei Kosut, David M. Kristol, Daniel LaLiberte, Ben Laurie, Paul J. Leach, Albert Lunde, John C. Mallery, Jean-Philippe Martin-Flatin, Mitra, David Morris, Gavin Nicol, Ross Patterson, Bill Perry, Jeffrey Perry, Scott Powers, Owen Rees, Luigi Rizzo, David Robinson, Marc Salomon, Rich Salz, Allan M. Schiffman, Jim Seidman, Chuck Shotton, Eric W. Sink, Simon E. Spero, Richard N. Taylor, Robert S. Thau, Bill (BearHeart) Weinman, Francois Yergeau, Mary Ellen Zurko.

Thanks to the "cave men" of Palo Alto. You know who you are.

Jim Gettys (the editor of [RFC2616]) wishes particularly to thank Roy Fielding, the editor of [RFC2068], along with John Klensin, Jeff Mogul, Paul Leach, Dave Kristol, Koen Holtman, John Franks, Josh Cohen, Alex Hopmann, Scott Lawrence, and Larry Masinter for their help. And thanks go particularly to Jeff Mogul and Scott Lawrence for performing the "MUST/MAY/SHOULD" audit.

The Apache Group, Anselm Baird-Smith, author of Jigsaw, and Henrik Frystyk implemented RFC 2068 early, and we wish to thank them for the discovery of many of the problems that this document attempts to rectify.

This specification makes heavy use of the augmented BNF and generic constructs defined by David H. Crocker for [RFC5234]. Similarly, it reuses many of the definitions provided by Nathaniel Borenstein and Ned Freed for MIME [RFC2045]. We hope that their inclusion in this specification will help reduce past confusion over the relationship between HTTP and Internet mail message formats.

13. References

13.1. Normative References

[ISO-8859-1]	International Organization for Standardization, "Information technology -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1 ", ISO/IEC 8859-1:1998, 1998.
[Part2]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y. and J. F. Reschke, "HTTP/1.1, part 2: Message Semantics", Internet-Draft draft-ietf-httpbis-p2-semantics-13, March 2011.
[Part3]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y. and J. F. Reschke, "HTTP/1.1, part 3: Message Payload and Content Negotiation", Internet-Draft draft-ietf-httpbis-p3-payload-13, March 2011.
[Part6]	

	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Nottingham, M. and J. F. Reschke, "HTTP/1.1, part 6: Caching", Internet-Draft draft-ietf-httpbis-p6-cache-13, March 2011.
[RFC5234]	Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[RFC3986]	Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
[USASCII]	American National Standards Institute, "Coded Character Set -- 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.
[RFC1950]	Deutsch, L.P. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, May 1996. RFC 1950 is an Informational RFC, thus it might be less stable than this specification. On the other hand, this downward reference was present since the publication of RFC 2068 in 1997 (
[RFC1951]	Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996. RFC 1951 is an Informational RFC, thus it might be less stable than this specification. On the other hand, this downward reference was present since the publication of RFC 2068 in 1997 (
[RFC1952]	Deutsch, P., Gailly, J-L., Adler, M., Deutsch, L.P. and G. Randers-Pehrson, "GZIP file format specification version 4.3", RFC 1952, May 1996. RFC 1952 is an Informational RFC, thus it might be less stable than this specification. On the other hand, this downward reference was present since the publication of RFC 2068 in 1997 (

13.2. Informative References

[Nie1997]	Frystyk, H., Gettys, J., Prud'hommeaux, E., Lie, H. and C. Lilley, "Network Performance Effects of HTTP/1.1, CSS1, and PNG", ACM Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication SIGCOMM '97, September 1997.
[Pad1995]	Padmanabhan, V.N. and J.C. Mogul, "Improving HTTP Latency", Computer Networks and ISDN Systems v. 28, pp. 25-35, December 1995.

[RFC1123]	Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.
[RFC1900]	Carpenter, B. and Y. Rekhter, "Renumbering Needs Work", RFC 1900, February 1996.
[RFC1919]	Chatel, M., "Classical versus Transparent IP Proxies", RFC 1919, March 1996.
[RFC1945]	Berners-Lee, T., Fielding, R.T. and H.F. Nielsen, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.
[RFC2045]	Freed, N. and N.S. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.
[RFC2047]	Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.
[RFC2068]	Fielding, R., Gettys, J., Mogul, J., Nielsen, H. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997.
[RFC2145]	Mogul, J.C., Fielding, R.T., Gettys, J. and H.F. Nielsen, "Use and Interpretation of HTTP Version Numbers", RFC 2145, May 1997.
[RFC2616]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
[RFC2817]	Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", RFC 2817, May 2000.
[RFC2818]	Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
[RFC2965]	Kristol, D. M. and L. Montulli, "HTTP State Management Mechanism", RFC 2965, October 2000.
[RFC3040]	Cooper, I., Melve, I. and G. Tomlinson, "Internet Web Replication and Caching Taxonomy", RFC 3040, January 2001.
[RFC3864]	Klyne, G., Nottingham, M. and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.
[RFC4288]	Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.
[RFC4395]	Hansen, T., Hardie, T. and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 115, RFC 4395, February 2006.
[RFC5226]	Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
[RFC5322]	Resnick, P., "Internet Message Format", RFC 5322, October 2008.

[draft-ietf-httpstate-cookie]	Barth, A., "HTTP State Management Mechanism", Internet-Draft draft-ietf-httpstate-cookie-23, March 2011.
[BCP97]	Klensin, J. and S. Hartman, "Handling Normative References to Standards-Track Documents", BCP 97, RFC 4897, June 2007.
[Kri2001]	Kristol, D., "HTTP Cookies: Standards, Privacy, and Politics", ACM Transactions on Internet Technology Vol. 1, #2, November 2001.
[Spe]	Spero, S., "Analysis of HTTP Performance Problems", .
[Tou1998]	Touch, J., Heidemann, J. and K. Obraczka, "Analysis of HTTP Performance", ISI Research Report ISI/RR-98-463, Aug 1998. (original report dated Aug. 1996)

Appendix A. Tolerant Applications

Although this document specifies the requirements for the generation of HTTP/1.1 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.

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

The character encoding of a representation SHOULD be labeled as the lowest common denominator of the character codes used within that representation, with the exception that not labeling the representation is preferred over labeling the representation with the labels US-ASCII or ISO-8859-1. See [Part3].

Additional rules for requirements on parsing and encoding of dates and other potential problems with date encodings include:

- HTTP/1.1 clients and caches SHOULD assume that an RFC-850 date which appears to be more than 50 years in the future is in fact in the past (this helps solve the "year 2000" problem).

- Although all date formats are specified to be case-sensitive, recipients SHOULD match day, week and timezone names case-insensitively.

- An HTTP/1.1 implementation MAY internally represent a parsed Expires date as earlier than the proper value, but MUST NOT internally represent a parsed Expires date as later than the proper value.

*All expiration-related calculations MUST be done in GMT. The local time zone MUST NOT influence the calculation or comparison of an age or expiration time.

*If an HTTP header field incorrectly carries a date value with a time zone other than GMT, it MUST be converted into GMT using the most conservative possible conversion.

Appendix B. HTTP Version History

HTTP has been in use by the World-Wide Web global information initiative since 1990. The first version of HTTP, later referred to as HTTP/0.9, was a simple protocol for hypertext data transfer across the Internet with only a single request method (GET) and no metadata. HTTP/1.0, as defined by [RFC1945], added a range of request methods and MIME-like messaging that could include metadata about the data transferred and modifiers on the request/response semantics. However, HTTP/1.0 did not sufficiently take into consideration the effects of hierarchical proxies, caching, the need for persistent connections, or name-based virtual hosts. The proliferation of incompletely-implemented applications calling themselves "HTTP/1.0" further necessitated a protocol version change in order for two communicating applications to determine each other's true capabilities.

HTTP/1.1 remains compatible with HTTP/1.0 by including more stringent requirements that enable reliable implementations, adding only those new features that will either be safely ignored by an HTTP/1.0 recipient or only sent when communicating with a party advertising compliance with HTTP/1.1.

It is beyond the scope of a protocol specification to mandate compliance with previous versions. HTTP/1.1 was deliberately designed, however, to make supporting previous versions easy. We would expect a general-purpose HTTP/1.1 server to understand any valid request in the format of HTTP/1.0 and respond appropriately with an HTTP/1.1 message that only uses features understood (or safely ignored) by HTTP/1.0 clients. Likewise, would expect an HTTP/1.1 client to understand any valid HTTP/1.0 response.

Since HTTP/0.9 did not support header fields in a request, there is no mechanism for it to support name-based virtual hosts (selection of resource by inspection of the Host header field). Any server that implements name-based virtual hosts ought to disable support for HTTP/0.9. Most requests that appear to be HTTP/0.9 are, in fact, badly constructed HTTP/1.x requests wherein a buggy client failed to properly encode linear whitespace found in a URI reference and placed in the request-target.

Appendix B.1. Changes from HTTP/1.0

This section summarizes major differences between versions HTTP/1.0 and HTTP/1.1.

Appendix B.1.1. Multi-homed Web Servers

The requirements that clients and servers support the Host header field (Section 9.4), report an error if it is missing from an HTTP/1.1 request, and accept absolute URIs (Section 4.1.2) are among the most important changes defined by HTTP/1.1.
Older HTTP/1.0 clients assumed a one-to-one relationship of IP addresses and servers; there was no other established mechanism for distinguishing the intended server of a request than the IP address to which that request was directed. The Host header field was introduced during the development of HTTP/1.1 and, though it was quickly implemented by most HTTP/1.0 browsers, additional requirements were placed on all HTTP/1.1 requests in order to ensure complete adoption. At the time of this writing, most HTTP-based services are dependent upon the Host header field for targeting requests.

Appendix B.1.2. Keep-Alive Connections

For most implementations of HTTP/1.0, each connection is established by the client prior to the request and closed by the server after sending the response. However, some implementations implement the Keep-Alive version of persistent connections described in Section 19.7.1 of [RFC2068].
Some clients and servers might wish to be compatible with some previous implementations of persistent connections in HTTP/1.0 clients and servers. Persistent connections in HTTP/1.0 are explicitly negotiated as they are not the default behavior. HTTP/1.0 experimental implementations of persistent connections are faulty, and the new facilities in HTTP/1.1 are designed to rectify these problems. The problem was that some existing HTTP/1.0 clients might send Keep-Alive to a proxy server that doesn't understand Connection, which would then erroneously forward it to the next inbound server, which would establish the Keep-Alive connection and result in a hung HTTP/1.0 proxy waiting for the close on the response. The result is that HTTP/1.0 clients must be prevented from using Keep-Alive when talking to proxies.
However, talking to proxies is the most important use of persistent connections, so that prohibition is clearly unacceptable. Therefore, we need some other mechanism for indicating a persistent connection is desired, which is safe to use even when talking to an old proxy that ignores Connection. Persistent connections are the default for HTTP/1.1 messages; we introduce a new keyword (Connection: close) for declaring non-persistence. See Section 9.1.

Appendix B.2. Changes from RFC 2616

Empty list elements in list productions have been deprecated. (Section 1.2.1)

Rules about implicit linear whitespace between certain grammar productions have been removed; now it's only allowed when specifically pointed out in the ABNF. The NUL octet is no longer allowed in comment and quoted-string text. The quoted-pair rule no longer allows escaping control characters other than HTAB. Non-ASCII content in header fields and reason phrase has been obsoleted and made opaque (the TEXT rule was removed) (Section 1.2.2)

Clarify that HTTP-Version is case sensitive. (Section 2.5)

Require that invalid whitespace around field-names be rejected. (Section 3.2)

Require recipients to handle bogus Content-Length header fields as errors. (Section 3.3)

Remove reference to non-existent identity transfer-coding value tokens. (Sections 3.3 and 6.2)

Update use of abs_path production from RFC 1808 to the path-absolute + query components of RFC 3986. State that the asterisk form is allowed for the OPTIONS request method only. (Section 4.1.2)

Clarification that the chunk length does not include the count of the octets in the chunk header and trailer. Furthermore disallowed line folding in chunk extensions. (Section 6.2.1)

Remove hard limit of two connections per server. (Section 7.1.4)

Clarify exactly when close connection options must be sent. (Section 9.1)

Define the semantics of the "Upgrade" header field in responses other than 101 (this was incorporated from [RFC2817]). (Section 9.8)

Appendix C. Collected ABNF

```
BWS = OWS

Chunked-Body = *chunk last-chunk trailer-part CRLF
Connection = "Connection:" OWS Connection-v
Connection-v = *( "," OWS ) connection-token *( OWS "," [ OWS
 connection-token ] )
Content-Length = "Content-Length:" OWS 1*Content-Length-v
Content-Length-v = 1*DIGIT

Date = "Date:" OWS Date-v
Date-v = HTTP-date

GMT = %x47.4D.54 ; GMT

HTTP-Prot-Name = %x48.54.54.50 ; HTTP
HTTP-Version = HTTP-Prot-Name "/" 1*DIGIT "." 1*DIGIT
HTTP-date = rfc1123-date / obs-date
HTTP-message = start-line *( header-field CRLF ) CRLF [ message-body
 ]
Host = "Host:" OWS Host-v
Host-v = uri-host [ ":" port ]

Method = token

OWS = *( [ obs-fold ] WSP )

RWS = 1*( [ obs-fold ] WSP )
Reason-Phrase = *( WSP / VCHAR / obs-text )
Request = Request-Line *( header-field CRLF ) CRLF [ message-body ]
Request-Line = Method SP request-target SP HTTP-Version CRLF
Response = Status-Line *( header-field CRLF ) CRLF [ message-body ]

Status-Code = 3DIGIT
Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

TE = "TE:" OWS TE-v
TE-v = [ ( "," / t-codings ) *( OWS "," [ OWS t-codings ] ) ]
Trailer = "Trailer:" OWS Trailer-v
Trailer-v = *( "," OWS ) field-name *( OWS "," [ OWS field-name ] )
Transfer-Encoding = "Transfer-Encoding:" OWS Transfer-Encoding-v
Transfer-Encoding-v = *( "," OWS ) transfer-coding *( OWS "," [ OWS
 transfer-coding ] )

URI-reference = <URI-reference, defined in [RFC3986], Section 4.1>
Upgrade = "Upgrade:" OWS Upgrade-v
Upgrade-v = *( "," OWS ) product *( OWS "," [ OWS product ] )

Via = "Via:" OWS Via-v
Via-v = *( "," OWS ) received-protocol RWS received-by [ RWS comment
 ] *( OWS "," [ OWS received-protocol RWS received-by [ RWS comment ]
```

```
 ] )

absolute-URI = <absolute-URI, defined in [RFC3986], Section 4.3>
asctime-date = day-name SP date3 SP time-of-day SP year
attribute = token
authority = <authority, defined in [RFC3986], Section 3.2>

chunk = chunk-size *WSP [ chunk-ext ] CRLF chunk-data CRLF
chunk-data = 1*OCTET
chunk-ext = *( ";" *WSP chunk-ext-name [ "=" chunk-ext-val ] *WSP )
chunk-ext-name = token
chunk-ext-val = token / quoted-str-nf
chunk-size = 1*HEXDIG
comment = "(" *( ctext / quoted-cpair / comment ) ")"
connection-token = token
ctext = OWS / %x21-27 ; '!'-'''
 / %x2A-5B ; '*'-'['
 / %x5D-7E ; ']'-'~'
 / obs-text

date1 = day SP month SP year
date2 = day "-" month "-" 2DIGIT
date3 = month SP ( 2DIGIT / ( SP DIGIT ) )
day = 2DIGIT
day-name = %x4D.6F.6E ; Mon
 / %x54.75.65 ; Tue
 / %x57.65.64 ; Wed
 / %x54.68.75 ; Thu
 / %x46.72.69 ; Fri
 / %x53.61.74 ; Sat
 / %x53.75.6E ; Sun
day-name-l = %x4D.6F.6E.64.61.79 ; Monday
 / %x54.75.65.73.64.61.79 ; Tuesday
 / %x57.65.64.6E.65.73.64.61.79 ; Wednesday
 / %x54.68.75.72.73.64.61.79 ; Thursday
 / %x46.72.69.64.61.79 ; Friday
 / %x53.61.74.75.72.64.61.79 ; Saturday
 / %x53.75.6E.64.61.79 ; Sunday

field-content = *( WSP / VCHAR / obs-text )
field-name = token
field-value = *( field-content / OWS )

header-field = field-name ":" OWS [ field-value ] OWS
hour = 2DIGIT
http-URI = "http://" authority path-abempty [ "?" query ]
https-URI = "https://" authority path-abempty [ "?" query ]

last-chunk = 1*"0" *WSP [ chunk-ext ] CRLF
```

```
message-body = *OCTET
minute = 2DIGIT
month = %x4A.61.6E ; Jan
 / %x46.65.62 ; Feb
 / %x4D.61.72 ; Mar
 / %x41.70.72 ; Apr
 / %x4D.61.79 ; May
 / %x4A.75.6E ; Jun
 / %x4A.75.6C ; Jul
 / %x41.75.67 ; Aug
 / %x53.65.70 ; Sep
 / %x4F.63.74 ; Oct
 / %x4E.6F.76 ; Nov
 / %x44.65.63 ; Dec

obs-date = rfc850-date / asctime-date
obs-fold = CRLF
obs-text = %x80-FF

partial-URI = relative-part [ "?" query ]
path-abempty = <path-abempty, defined in [RFC3986], Section 3.3>
path-absolute = <path-absolute, defined in [RFC3986], Section 3.3>
port = <port, defined in [RFC3986], Section 3.2.3>
product = token [ "/" product-version ]
product-version = token
protocol-name = token
protocol-version = token
pseudonym = token

qdtext = OWS / "!" / %x23-5B ; '#'-'['
 / %x5D-7E ; ']'-'~'
 / obs-text
qdtext-nf = WSP / "!" / %x23-5B ; '#'-'['
 / %x5D-7E ; ']'-'~'
 / obs-text
query = <query, defined in [RFC3986], Section 3.4>
quoted-cpair = "\" ( WSP / VCHAR / obs-text )
quoted-pair = "\" ( WSP / VCHAR / obs-text )
quoted-str-nf = DQUOTE *( qdtext-nf / quoted-pair ) DQUOTE
quoted-string = DQUOTE *( qdtext / quoted-pair ) DQUOTE
qvalue = ( "0" [ "." *3DIGIT ] ) / ( "1" [ "." *3"0" ] )

received-by = ( uri-host [ ":" port ] ) / pseudonym
received-protocol = [ protocol-name "/" ] protocol-version
relative-part = <relative-part, defined in [RFC3986], Section 4.2>
request-target = "*" / absolute-URI / ( path-absolute [ "?" query ] )
 / authority
rfc1123-date = day-name "," SP date1 SP time-of-day SP GMT
```

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

second = 2DIGIT
special = "(" / ")" / "<" / ">" / "@" / "," / ";" / ":" / "\" /
 DQUOTE / "/" / "[" / "]" / "?" / "=" / "{" / "}"
start-line = Request-Line / Status-Line

t-codings = "trailers" / ( transfer-extension [ te-params ] )
tchar = "!" / "#" / "$" / "%" / "&" / "'" / "*" / "+" / "-" / "." /
 "^" / "_" / "`" / "|" / "~" / DIGIT / ALPHA
te-ext = OWS ";" OWS token [ "=" word ]
te-params = OWS ";" OWS "q=" qvalue *te-ext
time-of-day = hour ":" minute ":" second
token = 1*tchar
trailer-part = *( header-field CRLF )
transfer-coding = "chunked" / "compress" / "deflate" / "gzip" /
 transfer-extension
transfer-extension = token *( OWS ";" OWS transfer-parameter )
transfer-parameter = attribute BWS "=" BWS value

uri-host = <host, defined in [RFC3986], Section 3.2.2>

value = word

word = token / quoted-string

year = 4DIGIT

ABNF diagnostics:

; Chunked-Body defined but not used
; Connection defined but not used
; Content-Length defined but not used
; Date defined but not used
; HTTP-message defined but not used
; Host defined but not used
; Request defined but not used
; Response defined but not used
; TE defined but not used
; Trailer defined but not used
; Transfer-Encoding defined but not used
; URI-reference defined but not used
; Upgrade defined but not used
; Via defined but not used
; http-URI defined but not used
; https-URI defined but not used
; partial-URI defined but not used
; special defined but not used
```

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

Appendix D.1. Since RFC 2616

Extracted relevant partitions from [RFC2616].

Appendix D.2. Since draft-ietf-httpbis-p1-messaging-00

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/1: "HTTP Version should be case sensitive" (http://purl.org/NET/http-errata#verscase)

* http://tools.ietf.org/wg/httpbis/trac/ticket/2: "'unsafe' characters" (http://purl.org/NET/http-errata#unsafe-uri)

* http://tools.ietf.org/wg/httpbis/trac/ticket/3: "Chunk Size Definition" (http://purl.org/NET/http-errata#chunk-size)

* http://tools.ietf.org/wg/httpbis/trac/ticket/4: "Message Length" (http://purl.org/NET/http-errata#msg-len-chars)

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

* http://tools.ietf.org/wg/httpbis/trac/ticket/11: "URI includes query" (http://purl.org/NET/http-errata#uriquery)

* http://tools.ietf.org/wg/httpbis/trac/ticket/15: "No close on 1xx responses" (http://purl.org/NET/http-errata#noclose1xx)

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

* http://tools.ietf.org/wg/httpbis/trac/ticket/26: "Import query BNF"

* http://tools.ietf.org/wg/httpbis/trac/ticket/31: "qdtext BNF"

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

* http://tools.ietf.org/wg/httpbis/trac/ticket/42: "RFC2606 Compliance"

* http://tools.ietf.org/wg/httpbis/trac/ticket/45: "RFC977 reference"

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

* http://tools.ietf.org/wg/httpbis/trac/ticket/47: "inconsistency in date format explanation"

* http://tools.ietf.org/wg/httpbis/trac/ticket/48: "Date reference typo"

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

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

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

Other changes:

* Update media type registrations to use RFC4288 template.

* Use names of RFC4234 core rules DQUOTE and WSP, fix broken ABNF for chunk-data (work in progress on http://tools.ietf.org/wg/httpbis/trac/ticket/36)

Appendix D.3. Since draft-ietf-httpbis-p1-messaging-01

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/19: "Bodies on GET (and other) requests"

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

* http://tools.ietf.org/wg/httpbis/trac/ticket/57: "Status Code and Reason Phrase"

* http://tools.ietf.org/wg/httpbis/trac/ticket/82: "rel_path not used"

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

* Get rid of duplicate BNF rule names ("host" -> "uri-host", "trailer" -> "trailer-part").

* Avoid underscore character in rule names ("http_URL" -> "http-URL", "abs_path" -> "path-absolute").

*Add rules for terms imported from URI spec ("absoluteURI", "authority", "path-absolute", "port", "query", "relativeURI", "host) — these will have to be updated when switching over to RFC3986.

*Synchronize core rules with RFC5234.

*Get rid of prose rules that span multiple lines.

*Get rid of unused rules LOALPHA and UPALPHA.

*Move "Product Tokens" section (back) into Part 1, as "token" is used in the definition of the Upgrade header field.

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

*Rewrite prose rule "token" in terms of "tchar", rewrite prose rule "TEXT".

Appendix D.4. Since draft-ietf-httpbis-p1-messaging-02

Closed issues:

*http://tools.ietf.org/wg/httpbis/trac/ticket/51: "HTTP-date vs. rfc1123-date"

*http://tools.ietf.org/wg/httpbis/trac/ticket/64: "WS in quoted-pair"

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

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

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

*Replace string literals when the string really is case-sensitive (HTTP-Version).

Appendix D.5. Since draft-ietf-httpbis-p1-messaging-03

Closed issues:

*http://tools.ietf.org/wg/httpbis/trac/ticket/28: "Connection closing"

* http://tools.ietf.org/wg/httpbis/trac/ticket/97: "Move registrations and registry information to IANA Considerations"

* http://tools.ietf.org/wg/httpbis/trac/ticket/120: "need new URL for PAD1995 reference"

* http://tools.ietf.org/wg/httpbis/trac/ticket/127: "IANA Considerations: update HTTP URI scheme registration"

* http://tools.ietf.org/wg/httpbis/trac/ticket/128: "Cite HTTPS URI scheme definition"

* http://tools.ietf.org/wg/httpbis/trac/ticket/129: "List-type headers vs Set-Cookie"

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

* Replace string literals when the string really is case-sensitive (HTTP-Date).

* Replace HEX by HEXDIG for future consistence with RFC 5234's core rules.

Appendix D.6. Since draft-ietf-httpbis-p1-messaging-04

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/34: "Out-of-date reference for URIs"

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

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

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

* Get rid of RFC822 dependency; use RFC5234 plus extensions instead.

* Only reference RFC 5234's core rules.

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

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

Appendix D.7. Since draft-ietf-httpbis-p1-messaging-05

Closed issues:

- *http://tools.ietf.org/wg/httpbis/trac/ticket/30: "Header LWS"
- *http://tools.ietf.org/wg/httpbis/trac/ticket/52: "Sort 1.3 Terminology"
- *http://tools.ietf.org/wg/httpbis/trac/ticket/63: "RFC2047 encoded words"
- *http://tools.ietf.org/wg/httpbis/trac/ticket/74: "Character Encodings in TEXT"
- *http://tools.ietf.org/wg/httpbis/trac/ticket/77: "Line Folding"
- *http://tools.ietf.org/wg/httpbis/trac/ticket/83: "OPTIONS * and proxies"
- *http://tools.ietf.org/wg/httpbis/trac/ticket/94: "Reason-Phrase BNF"
- *http://tools.ietf.org/wg/httpbis/trac/ticket/111: "Use of TEXT"
- *http://tools.ietf.org/wg/httpbis/trac/ticket/118: "Join "Differences Between HTTP Entities and RFC 2045 Entities"?"
- *http://tools.ietf.org/wg/httpbis/trac/ticket/134: "RFC822 reference left in discussion of date formats"

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

- *Rewrite definition of list rules, deprecate empty list elements.
- *Add appendix containing collected and expanded ABNF.

Other changes:

- *Rewrite introduction; add mostly new Architecture Section.
- *Move definition of quality values from Part 3 into Part 1; make TE request header field grammar independent of accept-params (defined in Part 3).

Appendix D.8. Since draft-ietf-httpbis-p1-messaging-06

Closed issues:

- http://tools.ietf.org/wg/httpbis/trac/ticket/161: "base for numeric protocol elements"
- http://tools.ietf.org/wg/httpbis/trac/ticket/162: "comment ABNF"

Partly resolved issues:

- http://tools.ietf.org/wg/httpbis/trac/ticket/88: "205 Bodies" (took out language that implied that there might be methods for which a request body MUST NOT be included)
- http://tools.ietf.org/wg/httpbis/trac/ticket/163: "editorial improvements around HTTP-date"

Appendix D.9. Since draft-ietf-httpbis-p1-messaging-07

Closed issues:

- http://tools.ietf.org/wg/httpbis/trac/ticket/93: "Repeating single-value headers"
- http://tools.ietf.org/wg/httpbis/trac/ticket/131: "increase connection limit"
- http://tools.ietf.org/wg/httpbis/trac/ticket/157: "IP addresses in URLs"
- http://tools.ietf.org/wg/httpbis/trac/ticket/172: "take over HTTP Upgrade Token Registry"
- http://tools.ietf.org/wg/httpbis/trac/ticket/173: "CR and LF in chunk extension values"
- http://tools.ietf.org/wg/httpbis/trac/ticket/184: "HTTP/0.9 support"
- http://tools.ietf.org/wg/httpbis/trac/ticket/188: "pick IANA policy (RFC5226) for Transfer Coding / Content Coding"
- http://tools.ietf.org/wg/httpbis/trac/ticket/189: "move definitions of gzip/deflate/compress to part 1"
- http://tools.ietf.org/wg/httpbis/trac/ticket/194: "disallow control characters in quoted-pair"

Partly resolved issues:

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

Appendix D.10. Since draft-ietf-httpbis-p1-messaging-08

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/201: "header parsing, treatment of leading and trailing OWS"

Partly resolved issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/60: "Placement of 13.5.1 and 13.5.2"

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

Appendix D.11. Since draft-ietf-httpbis-p1-messaging-09

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/73: "Clarification of the term 'deflate'"

* http://tools.ietf.org/wg/httpbis/trac/ticket/83: "OPTIONS * and proxies"

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

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

* http://tools.ietf.org/wg/httpbis/trac/ticket/165: "Case-sensitivity of HTTP-date"

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

Partly resolved issues:

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

Appendix D.12. Since draft-ietf-httpbis-p1-messaging-10

Closed issues:

- http://tools.ietf.org/wg/httpbis/trac/ticket/28: "Connection Closing"
- http://tools.ietf.org/wg/httpbis/trac/ticket/90: "Delimiting messages with multipart/byteranges"
- http://tools.ietf.org/wg/httpbis/trac/ticket/95: "Handling multiple Content-Length headers"
- http://tools.ietf.org/wg/httpbis/trac/ticket/109: "Clarify entity / representation / variant terminology"
- http://tools.ietf.org/wg/httpbis/trac/ticket/220: "consider removing the 'changes from 2068' sections"

Partly resolved issues:

- http://tools.ietf.org/wg/httpbis/trac/ticket/159: "HTTP(s) URI scheme definitions"

Appendix D.13. Since draft-ietf-httpbis-p1-messaging-11

Closed issues:

- http://tools.ietf.org/wg/httpbis/trac/ticket/193: "Trailer requirements"
- http://tools.ietf.org/wg/httpbis/trac/ticket/204: "Text about clock requirement for caches belongs in p6"
- http://tools.ietf.org/wg/httpbis/trac/ticket/221: "effective request URI: handling of missing host in HTTP/1.0"
- http://tools.ietf.org/wg/httpbis/trac/ticket/248: "confusing Date requirements for clients"

Partly resolved issues:

- http://tools.ietf.org/wg/httpbis/trac/ticket/95: "Handling multiple Content-Length headers"

Appendix D.14. Since draft-ietf-httpbis-p1-messaging-12

Closed issues:

- http://tools.ietf.org/wg/httpbis/trac/ticket/75: "RFC2145 Normative"
- http://tools.ietf.org/wg/httpbis/trac/ticket/159: "HTTP(s) URI scheme definitions" (tune the requirements on userinfo)
- http://tools.ietf.org/wg/httpbis/trac/ticket/210: "define 'transparent' proxy"
- http://tools.ietf.org/wg/httpbis/trac/ticket/224: "Header Classification"
- http://tools.ietf.org/wg/httpbis/trac/ticket/233: "Is * usable as a request-uri for new methods?"
- http://tools.ietf.org/wg/httpbis/trac/ticket/240: "Migrate Upgrade details from RFC2817"
- http://tools.ietf.org/wg/httpbis/trac/ticket/276: "untangle ABNFs for header fields"
- http://tools.ietf.org/wg/httpbis/trac/ticket/279: "update RFC 2109 reference"

Index

	outbound
P	
	proxy
R	
	reverse proxy
T	
	transforming proxy
	transparent proxy
	tunnel
U	
	upstream

Authors' Addresses

Roy T. Fielding editor Fielding Adobe Systems Incorporated 345 Park Ave San Jose, CA 95110 USA EMail: fielding@gbiv.com URI: http://roy.gbiv.com/

Jim Gettys Gettys Alcatel-Lucent Bell Labs 21 Oak Knoll Road Carlisle, MA 01741 USA EMail: jg@freedesktop.org URI: http://gettys.wordpress.com/

Jeffrey C. Mogul Mogul Hewlett-Packard Company HP Labs, Large Scale Systems Group 1501 Page Mill Road, MS 1177 Palo Alto, CA 94304 USA EMail: JeffMogul@acm.org

Henrik Frystyk Nielsen Frystyk Microsoft Corporation
1 Microsoft Way Redmond, WA 98052 USA EMail: henrikn@microsoft.com

Larry Masinter Masinter Adobe Systems Incorporated 345 Park Ave San Jose, CA 95110 USA EMail: LMM@acm.org URI: http://larry.masinter.net/

Paul J. Leach Leach Microsoft Corporation 1 Microsoft Way Redmond, WA 98052 EMail: paulle@microsoft.com

Tim Berners-Lee Berners-Lee World Wide Web Consortium MIT Computer Science and Artificial Intelligence Laboratory The Stata Center, Building 32 32 Vassar Street Cambridge, MA 02139 USA EMail: timbl@w3.org URI: http://www.w3.org/People/Berners-Lee/

Yves Lafon editor Lafon World Wide Web Consortium W3C / ERCIM 2004, rte des Lucioles Sophia-Antipolis, AM 06902 France EMail: ylafon@w3.org URI: http://www.raubacapeu.net/people/yves/

Julian F. Reschke editor Reschke greenbytes GmbH Hafenweg 16 Muenster, NW 48155 Germany Phone: +49 251 2807760 EMail: julian.reschke@greenbytes.de URI: http://greenbytes.de/tech/webdav/