

HTTPbis Working Group
Internet-Draft
Intended status: Informational
Expires: August 23, 2012

J. Reschke
greenbytes
February 20, 2012

Initial Hypertext Transfer Protocol (HTTP)
Authentication Scheme Registrations
draft-ietf-httpbis-authscheme-registrations-03

Abstract

This document registers Hypertext Transfer Protocol (HTTP) authentication schemes which have been defined in standards-track RFCs before the IANA HTTP Authentication Scheme Registry was established.

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), which is archived at <http://lists.w3.org/Archives/Public/ietf-http-wg/>.

The current issues list is at <http://trac.tools.ietf.org/wg/httpbis/trac/query?component=authscheme-registrations> 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 B.3.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 23, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Security Considerations	3
3. IANA Considerations	3
4. Normative References	3
Appendix A. Initial Registry Contents	4
Appendix B. Change Log (to be removed by RFC Editor before publication)	4
B.1. Since draft-ietf-httpbis-authscheme-registrations-00 . . .	4
B.2. Since draft-ietf-httpbis-authscheme-registrations-01 . . .	4
B.3. Since draft-ietf-httpbis-authscheme-registrations-02 . . .	4

1. Introduction

This document registers Hypertext Transfer Protocol (HTTP) authentication schemes which have been defined in standards-track RFCs before the IANA HTTP Authentication Scheme Registry was established.

2. Security Considerations

There are no security considerations related to the registration itself.

3. IANA Considerations

Appendix A provides initial registrations of HTTP authentication schemes for the IANA HTTP Authentication Scheme registry at <http://www.iana.org/assignments/http-authschemes> (see Section 2.3 of [draft-ietf-httpbis-p7-auth]).

4. Normative References

- | | |
|------------------------------|---|
| [RFC2617] | Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999. |
| [RFC4559] | Jaganathan, K., Zhu, L., and J. Brezak, "SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows", RFC 4559, June 2006. |
| [draft-ietf-httpbis-p7-auth] | Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 7: Authentication", draft-ietf-httpbis-p7-auth-18 (work in progress), January 2012. |

Appendix A. Initial Registry Contents

Authentication Scheme Name	Reference	Notes
Basic	[RFC2617], Section 2	This authentication scheme violates both HTTP semantics (being connection-oriented) and syntax (use of syntax incompatible with the WWW-Authenticate and Authorization header field syntax).
Digest	[RFC2617], Section 3	
Negotiate	[RFC4559], Section 3	

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

B.1. Since draft-ietf-httpbis-authscheme-registrations-00

Update draft-ietf-httpbis-p7-auth reference.

B.2. Since draft-ietf-httpbis-authscheme-registrations-01

Update draft-ietf-httpbis-p7-auth reference.

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/308>>: "need to reserve 'negotiate' as auth scheme name"

B.3. Since draft-ietf-httpbis-authscheme-registrations-02

Update draft-ietf-httpbis-p7-auth reference.

Author's Address

Julian F. Reschke
greenbytes GmbH
Hafenweg 16
Muenster, NW 48155
Germany

EMail: julian.reschke@greenbytes.de
URI: <http://greenbytes.de/tech/webdav/>

HTTPbis Working Group
Internet-Draft
Intended status: Informational
Expires: January 3, 2013

J. Reschke
greenbytes
July 2, 2012

Initial Hypertext Transfer Protocol (HTTP) Method Registrations
draft-ietf-httpbis-method-registrations-08

Abstract

This document registers those Hypertext Transfer Protocol (HTTP) methods which have been defined in standards-track RFCs before the IANA HTTP Method Registry was established.

Editorial Note (To be removed by RFC Editor)

Discussion of this draft ought to take place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org), which is archived at <http://lists.w3.org/Archives/Public/ietf-http-wg/>.

The current issues list is at <http://trac.tools.ietf.org/wg/httpbis/trac/query?component=method-registrations> 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 B.8.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 3, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the

document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Security Considerations	3
3. IANA Considerations	3
4. Normative References	3
Appendix A. Initial Registry Contents	5
Appendix B. Change Log (to be removed by RFC Editor before publication)	5
B.1. Since draft-ietf-httpbis-method-registrations-00	5
B.2. Since draft-ietf-httpbis-method-registrations-01	6
B.3. Since draft-ietf-httpbis-method-registrations-02	6
B.4. Since draft-ietf-httpbis-method-registrations-03	6
B.5. Since draft-ietf-httpbis-method-registrations-04	6
B.6. Since draft-ietf-httpbis-method-registrations-05	6
B.7. Since draft-ietf-httpbis-method-registrations-06	6
B.8. Since draft-ietf-httpbis-method-registrations-07	6

1. Introduction

This document registers those Hypertext Transfer Protocol (HTTP) methods which have been defined in standards-track RFCs other than [draft-ietf-httpbis-p2-semantics] before the IANA HTTP Method Registry was established.

2. Security Considerations

There are no security considerations related to the registration itself.

3. IANA Considerations

Appendix A provides initial registrations of HTTP method names for the IANA HTTP Method registry at <http://www.iana.org/assignments/http-methods> (see Section 2.2 of [draft-ietf-httpbis-p2-semantics]).

4. Normative References

- | | |
|-----------|--|
| [RFC2068] | Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997. |
| [RFC3253] | Clemm, G., Amsden, J., Ellison, T., Kaler, C., and J. Whitehead, "Versioning Extensions to WebDAV (Web Distributed Authoring and Versioning)", RFC 3253, March 2002. |
| [RFC3648] | Whitehead, J. and J. Reschke, Ed., "Web Distributed Authoring and Versioning (WebDAV) Ordered Collections Protocol", RFC 3648, December 2003. |
| [RFC3744] | Clemm, G., Reschke, J., Sedlar, E., and J. Whitehead, "Web Distributed Authoring and Versioning (WebDAV) Access Control Protocol", RFC 3744, May 2004. |
| [RFC4437] | Whitehead, J., Clemm, G., and J. Reschke, Ed., "Web Distributed |

- Authoring and Versioning (WebDAV)
Redirect Reference Resources",
RFC 4437, March 2006.
- [RFC4791] Daboo, C., Desruisseaux, B., and
L. Dusseault, "Calendaring
Extensions to WebDAV (CalDAV)",
RFC 4791, March 2007.
- [RFC4918] Dusseault, L., Ed., "HTTP
Extensions for Web Distributed
Authoring and Versioning
(WebDAV)", RFC 4918, June 2007.
- [RFC5323] Reschke, J., Ed., Reddy, S.,
Davis, J., and A. Babich, "Web
Distributed Authoring and
Versioning (WebDAV) SEARCH",
RFC 5323, November 2008.
- [RFC5789] Dusseault, L. and J. Snell, "PATCH
Method for HTTP", RFC 5789,
March 2010.
- [RFC5842] Clemm, G., Crawford, J., Reschke,
J., Ed., and J. Whitehead,
"Binding Extensions to Web
Distributed Authoring and
Versioning (WebDAV)", RFC 5842,
April 2010.
- [draft-ietf-httpbis-p2-semantics] Fielding, R., Ed., Lafon, Y., Ed.,
and J. Reschke, Ed., "HTTP/1.1,
part 2: Message Semantics",
draft-ietf-httpbis-p2-semantics-19
(work in progress), March 2012.

Appendix A. Initial Registry Contents

Method Name	Safe	Reference
ACL	no	[RFC3744], Section 8.1
BASELINE-CONTROL	no	[RFC3253], Section 12.6
BIND	no	[RFC5842], Section 4
CHECKIN	no	[RFC3253], Section 4.4 and [RFC3253], Section 9.4
CHECKOUT	no	[RFC3253], Section 4.3 and [RFC3253], Section 8.8
COPY	no	[RFC4918], Section 9.8
LABEL	no	[RFC3253], Section 8.2
LINK	no	[RFC2068], Section 19.6.1.2
LOCK	no	[RFC4918], Section 9.10
MERGE	no	[RFC3253], Section 11.2
MKACTIVITY	no	[RFC3253], Section 13.5
MKCALENDAR	no	[RFC4791], Section 5.3.1
MKCOL	no	[RFC4918], Section 9.3
MKREDIRECTREF	no	[RFC4437], Section 6
MKWORKSPACE	no	[RFC3253], Section 6.3
MOVE	no	[RFC4918], Section 9.9
ORDERPATCH	no	[RFC3648], Section 7
PATCH	no	[RFC5789], Section 2
PROPFIND	yes	[RFC4918], Section 9.1
PROPPATCH	no	[RFC4918], Section 9.2
REBIND	no	[RFC5842], Section 6
REPORT	yes	[RFC3253], Section 3.6
SEARCH	yes	[RFC5323], Section 2
UNBIND	no	[RFC5842], Section 5
UNCHECKOUT	no	[RFC3253], Section 4.5
UNLINK	no	[RFC2068], Section 19.6.1.3
UNLOCK	no	[RFC4918], Section 9.11
UPDATE	no	[RFC3253], Section 7.1
UPDATEREDIRECTREF	no	[RFC4437], Section 7
VERSION-CONTROL	no	[RFC3253], Section 3.5

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

B.1. Since draft-ietf-httpbis-method-registrations-00

Added SEARCH method (RFC 5323).

B.2. Since draft-ietf-httpbis-method-registrations-01

Update draft-ietf-httpbis-p2-semantics reference.

B.3. Since draft-ietf-httpbis-method-registrations-02

Update draft-ietf-httpbis-p2-semantics reference. PATCH is now defined in draft-dusseault-http-patch. BIND, UNBIND and REBIND are defined in draft-ietf-webdav-bind. Drop the "updates draft-ietf-httpbis-p2-semantics" clause.

B.4. Since draft-ietf-httpbis-method-registrations-03

draft-dusseault-http-patch was published as RFC 5789. draft-ietf-webdav-bind was published as RFC 5842. Fix typo in MKACTIVITY entry. Update draft-ietf-httpbis-p2-semantics reference. Fix change log section titles.

B.5. Since draft-ietf-httpbis-method-registrations-04

Update draft-ietf-httpbis-p2-semantics reference.

B.6. Since draft-ietf-httpbis-method-registrations-05

Update draft-ietf-httpbis-p2-semantics reference.

B.7. Since draft-ietf-httpbis-method-registrations-06

Update draft-ietf-httpbis-p2-semantics reference.

B.8. Since draft-ietf-httpbis-method-registrations-07

Update draft-ietf-httpbis-p2-semantics reference.

Author's Address

Julian F. Reschke
greenbytes GmbH
Hafenweg 16
Muenster, NW 48155
Germany

EMail: julian.reschke@greenbytes.de
URI: <http://greenbytes.de/tech/webdav/>

HTTPbis Working Group
Internet-Draft
Obsoletes: 2145,2616 (if approved)
Updates: 2817 (if approved)
Intended status: Standards Track
Expires: January 17, 2013

R. Fielding, Ed.
Adobe
Y. Lafon, Ed.
W3C
J. Reschke, Ed.
greenbytes
July 16, 2012

HTTP/1.1, part 1: Message Routing and Syntax"
draft-ietf-httpbis-pl-messaging-20

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 provides an overview of HTTP architecture and its associated terminology, defines the "http" and "https" Uniform Resource Identifier (URI) schemes, defines the HTTP/1.1 message syntax and parsing requirements, and describes general security concerns for implementations.

Editorial Note (To be removed by RFC Editor)

Discussion of this draft takes place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org), which is archived at <http://lists.w3.org/Archives/Public/ietf-http-wg/>.

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 D.21.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 17, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	6
1.1. Requirement Notation	7
1.2. Syntax Notation	7
2. Architecture	7
2.1. Client/Server Messaging	7
2.2. Implementation Diversity	9
2.3. Connections and Transport Independence	10
2.4. Intermediaries	10
2.5. Caches	13
2.6. Conformance and Error Handling	13
2.7. Protocol Versioning	14
2.8. Uniform Resource Identifiers	16
2.8.1. http URI scheme	17
2.8.2. https URI scheme	18

2.8.3.	http and https URI Normalization and Comparison . . .	19
3.	Message Format	20
3.1.	Start Line	20
3.1.1.	Request Line	21
3.1.2.	Status Line	22
3.2.	Header Fields	23
3.2.1.	Whitespace	24
3.2.2.	Field Parsing	25
3.2.3.	Field Length	25
3.2.4.	Field value components	26
3.3.	Message Body	27
3.3.1.	Transfer-Encoding	27
3.3.2.	Content-Length	29
3.3.3.	Message Body Length	30
3.4.	Handling Incomplete Messages	32
3.5.	Message Parsing Robustness	33
4.	Transfer Codings	33
4.1.	Chunked Transfer Coding	34
4.2.	Compression Codings	36
4.2.1.	Compress Coding	36
4.2.2.	Deflate Coding	36
4.2.3.	Gzip Coding	36
4.3.	TE	36
4.3.1.	Quality Values	38
4.4.	Trailer	38
5.	Message Routing	39
5.1.	Identifying a Target Resource	39
5.2.	Connecting Inbound	39
5.3.	Request Target	40
5.4.	Host	42
5.5.	Effective Request URI	43
5.6.	Intermediary Forwarding	44
5.6.1.	End-to-end and Hop-by-hop Header Fields	45
5.6.2.	Non-modifiable Header Fields	46
5.7.	Associating a Response to a Request	47
6.	Connection Management	47
6.1.	Connection	47
6.2.	Via	49
6.3.	Persistent Connections	50
6.3.1.	Purpose	50
6.3.2.	Overall Operation	51
6.3.3.	Practical Considerations	53
6.3.4.	Retrying Requests	53
6.4.	Message Transmission Requirements	54
6.4.1.	Persistent Connections and Flow Control	54
6.4.2.	Monitoring Connections for Error Status Messages	54
6.4.3.	Use of the 100 (Continue) Status	54
6.4.4.	Closing Connections on Error	56

6.5. Upgrade	56
7. IANA Considerations	58
7.1. Header Field Registration	58
7.2. URI Scheme Registration	59
7.3. Internet Media Type Registrations	59
7.3.1. Internet Media Type message/http	59
7.3.2. Internet Media Type application/http	60
7.4. Transfer Coding Registry	61
7.5. Transfer Coding Registrations	62
7.6. Upgrade Token Registry	62
7.7. Upgrade Token Registration	63
8. Security Considerations	63
8.1. Personal Information	63
8.2. Abuse of Server Log Information	64
8.3. Attacks Based On File and Path Names	64
8.4. DNS-related Attacks	65
8.5. Intermediaries and Caching	65
8.6. Protocol Element Size Overflows	65
9. Acknowledgments	66
10. References	67
10.1. Normative References	67
10.2. Informative References	68
Appendix A. HTTP Version History	71
A.1. Changes from HTTP/1.0	71
A.1.1. Multi-homed Web Servers	72
A.1.2. Keep-Alive Connections	72
A.1.3. Introduction of Transfer-Encoding	73
A.2. Changes from RFC 2616	73
Appendix B. ABNF list extension: #rule	74
Appendix C. Collected ABNF	75
Appendix D. Change Log (to be removed by RFC Editor before publication)	78
D.1. Since RFC 2616	78
D.2. Since draft-ietf-httpbis-pl-messaging-00	78
D.3. Since draft-ietf-httpbis-pl-messaging-01	79
D.4. Since draft-ietf-httpbis-pl-messaging-02	80
D.5. Since draft-ietf-httpbis-pl-messaging-03	81
D.6. Since draft-ietf-httpbis-pl-messaging-04	81
D.7. Since draft-ietf-httpbis-pl-messaging-05	82
D.8. Since draft-ietf-httpbis-pl-messaging-06	83
D.9. Since draft-ietf-httpbis-pl-messaging-07	83
D.10. Since draft-ietf-httpbis-pl-messaging-08	84
D.11. Since draft-ietf-httpbis-pl-messaging-09	84
D.12. Since draft-ietf-httpbis-pl-messaging-10	85
D.13. Since draft-ietf-httpbis-pl-messaging-11	85
D.14. Since draft-ietf-httpbis-pl-messaging-12	86
D.15. Since draft-ietf-httpbis-pl-messaging-13	86
D.16. Since draft-ietf-httpbis-pl-messaging-14	87

D.17. Since draft-ietf-httpbis-pl-messaging-15	87
D.18. Since draft-ietf-httpbis-pl-messaging-16	87
D.19. Since draft-ietf-httpbis-pl-messaging-17	88
D.20. Since draft-ietf-httpbis-pl-messaging-18	88
D.21. Since draft-ietf-httpbis-pl-messaging-19	89
Index	89

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. This document is the first in a series of documents that collectively form the HTTP/1.1 specification:

RFC xxx1: Message Routing and Syntax

RFC xxx2: Semantics and Payloads

RFC xxx3: Conditional Requests

RFC xxx4: Range Requests

RFC xxx5: Caching

RFC xxx6: Authentication

This HTTP/1.1 specification obsoletes and moves to historic status RFC 2616, its predecessor RFC 2068, RFC 2145 (on HTTP versioning), and RFC 2817 (on using CONNECT for TLS upgrades).

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 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. Requirement Notation

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].

1.2. Syntax Notation

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] with the list rule extension defined in Appendix B. Appendix C shows the collected ABNF with the list rule expanded.

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

As a convention, ABNF rule names prefixed with "obs-" denote "obsolete" grammar rules that appear for historical reasons.

2. 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 (Section 3) 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.

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.

HTTP relies upon the Uniform Resource Identifier (URI) standard [RFC3986] to indicate the target resource (Section 5.1) 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 [Part2] for the differences between HTTP and MIME messages).

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).

```
      request    >
UA ===== O
      <    response
```

A client sends an HTTP request to a server in the form of a request message, beginning with a request-line that includes a method, URI, and protocol version (Section 3.1.1), followed by header fields containing request modifiers, client information, and representation metadata (Section 3.2), an empty line to indicate the end of the header section, and finally a message body containing the payload body (if any, Section 3.3).

A server responds to a client's request by sending one or more HTTP response messages, each beginning with a status line that includes the protocol version, a success or error code, and textual reason phrase (Section 3.1.2), possibly followed by header fields containing server information, resource metadata, and representation metadata (Section 3.2), an empty line to indicate the end of the header section, and finally a message body containing the payload body (if

any, Section 3.3).

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-Language: en, mi
```

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. Implementation Diversity

When considering the design of HTTP, it is easy to fall into a trap of thinking that all user agents are general-purpose browsers and all origin servers are large public websites. That is not the case in practice. Common HTTP user agents include household appliances, stereos, scales, software/firmware updaters, command-line programs, mobile apps, and communication devices in a multitude of shapes and sizes. Likewise, common HTTP origin servers include home automation units, configurable networking components, office machines, autonomous robots, news feeds, traffic cameras, ad selectors, and video delivery platforms.

The term "user agent" does not imply that there is a human user directly interacting with the software agent at the time of a request. In many cases, a user agent is installed or configured to run in the background and save its results for later inspection (or save only a subset of those results that might be interesting or erroneous). Spiders, for example, are typically given a start URI and configured to follow certain behavior while crawling the Web as a hypertext graph.

The implementation diversity of HTTP means that we cannot assume the user agent can make interactive suggestions to a user or provide adequate warning for security or privacy options. In the few cases where this specification requires reporting of errors to the user, it is acceptable for such reporting to only be visible in an error console or log file. Likewise, requirements that an automated action be confirmed by the user before proceeding can be met via advance configuration choices, run-time options, or simply not proceeding with the unsafe action.

2.3. Connections and Transport Independence

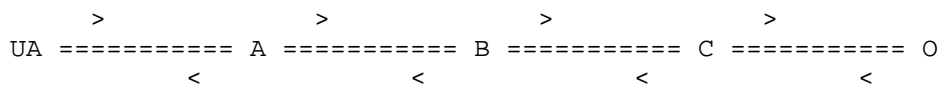
HTTP messaging is independent of the underlying transport or session-layer connection protocol(s). HTTP only presumes a reliable transport with in-order delivery of requests and the corresponding in-order delivery of responses. The mapping of HTTP request and response structures onto the data units of 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 URI (Section 5.1). For example, the "http" URI scheme (Section 2.8.1) 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 6.3.

2.4. 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. See Section 4.4.4 of [Part2] and Section 7.6 of [Part6] for status and warning codes related to transformations.

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** conform to HTTP user agent requirements on the gateway's inbound connection and **MUST** implement the Connection (Section 6.1) and Via (Section 6.2) 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.

The above categories for intermediary only consider those acting as participants in the HTTP communication. There are also intermediaries that can act on lower layers of the network protocol stack, filtering or redirecting HTTP traffic without the knowledge or permission of message senders. Network intermediaries often introduce security flaws or interoperability problems by violating HTTP semantics. For example, an "interception proxy" [RFC3040] (also commonly known as a "transparent proxy" [RFC1919] or "captive portal") differs from an HTTP proxy because it is not selected by the client. Instead, an interception proxy filters or redirects outgoing TCP port 80 packets (and occasionally other common port traffic). 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.

HTTP is defined as a stateless protocol, meaning that each request message can be understood in isolation. Many implementations depend on HTTP's stateless design in order to reuse proxied connections or dynamically load balance requests across multiple servers. Hence, servers **MUST NOT** assume that two requests on the same connection are from the same user agent unless the connection is secured and specific to that agent. Some non-standard HTTP extensions (e.g., [RFC4559]) have been known to violate this requirement, resulting in security and interoperability problems.

2.5. 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.

```

      >               >
UA ===== A ===== B - - - - - C - - - - - O
      <               <

```

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.

2.6. Conformance and Error Handling

This specification targets conformance criteria according to the role of a participant in HTTP communication. Hence, HTTP requirements are placed on senders, recipients, clients, servers, user agents, intermediaries, origin servers, proxies, gateways, or caches, depending on what behavior is being constrained by the requirement.

The verb "generate" is used instead of "send" where a requirement differentiates between creating a protocol element and merely forwarding a received element downstream.

An implementation is considered conformant if it complies with all of the requirements associated with the roles it partakes in HTTP. Note

that SHOULD-level requirements are relevant here, unless one of the documented exceptions is applicable.

In addition to the prose requirements placed upon them, senders MUST NOT generate protocol elements that do not match the grammar defined by the ABNF rules for those protocol elements that are applicable to the sender's role. If a received protocol element is processed, the recipient MUST be able to parse any value that would match the ABNF rules for that protocol element, excluding only those rules not applicable to the recipient's role.

Unless noted otherwise, a recipient MAY attempt to recover a usable protocol element from an invalid construct. HTTP does not define specific error handling mechanisms except when they have a direct impact on security, since different applications of the protocol require different error handling strategies. For example, a Web browser might wish to transparently recover from a response where the Location header field doesn't parse according to the ABNF, whereas a systems control client might consider any form of error recovery to be dangerous.

2.7. 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 conformance 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-name "/" DIGIT "." DIGIT
HTTP-name    = %x48.54.54.50 ; "HTTP", case-sensitive
```

The HTTP version number consists of two decimal digits separated by a "." (period or decimal point). The first digit ("major version") indicates the HTTP messaging syntax, whereas the second digit ("minor version") indicates the highest minor version to which the sender is conformant 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 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 conformant 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 a header field does not change between minor versions of the same major HTTP 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 conformance 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 6.1). These requirements allow HTTP's functionality to be enhanced without requiring prior update of deployed intermediaries.

Intermediaries that process HTTP messages (i.e., all intermediaries other than those acting as tunnels) **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 in that message matches a version to which that intermediary is conformant 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 to which the client is conformant 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 to which it is not conformant.

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 to which the server is conformant and whose major version is less than or equal to the one received in the request. An HTTP server MUST NOT send a version to which it is not conformant. 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 conform to 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.

2.8. 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 (Section 5.5).

2.8.1. 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 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 (Section 3) containing the URI's identifying data (Section 5) to the server. If the server responds to that request with a non-interim HTTP response message, as described in Section 4 of [Part2], 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.

2.8.2. 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 7.2 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 [RFC6265], 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].

2.8.3. 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.

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

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.

Recipients **MUST** parse an HTTP message as a sequence of octets in an encoding that is a superset of US-ASCII [USASCII]. Parsing an HTTP message as a stream of Unicode characters, without regard for the specific encoding, creates security vulnerabilities due to the varying ways that string processing libraries handle invalid multibyte character sequences that contain the octet LF (%x0A). String-based parsers can only be safely used within protocol elements after the element has been extracted from the message, such as within a header field-value after message parsing has delineated the individual fields.

An HTTP message can be parsed as a stream for incremental processing or forwarding downstream. However, recipients cannot rely on incremental delivery of partial messages, since some implementations will buffer or delay message forwarding for the sake of network efficiency, security checks, or payload transformations.

3.1. Start Line

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.

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.

3.1.1. Request Line

A 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

A server MUST be able to parse any received message that begins with a request-line and matches the ABNF rule for HTTP-message.

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

method = token

The methods defined by this specification can be found in Section 2 of [Part2], along with information regarding the HTTP method registry and considerations for defining new methods.

The request-target identifies the target resource upon which to apply the request, as defined in Section 5.3.

No whitespace is allowed inside the method, request-target, and protocol version. Hence, recipients typically parse the request-line into its component parts by splitting on the SP characters.

Unfortunately, some user agents fail to properly encode hypertext references that have embedded whitespace, sending the characters directly instead of properly percent-encoding the disallowed characters. Recipients of an invalid request-line SHOULD respond with either a 400 (Bad Request) error or a 301 (Moved Permanently) redirect with the request-target properly encoded. Recipients SHOULD

NOT attempt to autocorrect and then process the request without a redirect, since the invalid request-line might be deliberately crafted to bypass security filters along the request chain.

HTTP does not place a pre-defined limit on the length of a request-line. A server that receives a method longer than any that it implements SHOULD respond with either a 405 (Method Not Allowed), if it is an origin server, or a 501 (Not Implemented) status code. 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 4.6.12 of [Part2]).

Various ad-hoc limitations on request-line length are found in practice. It is RECOMMENDED that all HTTP senders and recipients support, at a minimum, request-line lengths of up to 8000 octets.

3.1.2. 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

A client MUST be able to parse any received message that begins with a status-line and matches the ABNF rule for HTTP-message.

The status-code element is a 3-digit integer code describing the result of the server's attempt to understand and satisfy the client's corresponding request. The rest of the response message is to be interpreted in light of the semantics defined for that status code. See Section 4 of [Part2] for information about the semantics of status codes, including the classes of status code (indicated by the first digit), the status codes defined by this specification, considerations for the definition of new status codes, and the IANA registry.

status-code = 3DIGIT

The reason-phrase element exists for the sole purpose of providing a textual description associated with the numeric status code, mostly out of deference to earlier Internet application protocols that were more frequently used with interactive text clients. A client SHOULD ignore the reason-phrase content.

reason-phrase = *(HTAB / SP / VCHAR / obs-text)

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 BWS
field-name      = token
field-value     = *( field-content / obs-fold )
field-content   = *( HTAB / SP / VCHAR / obs-text )
obs-fold       = CRLF ( SP / HTAB )
                ; obsolete line folding
                ; see Section 3.2.2
```

The field-name token labels the corresponding field-value as having the semantics defined by that header field. For example, the Date header field is defined in Section 9.10 of [Part2] as containing the origination timestamp for the message in which it appears.

HTTP header fields are fully extensible: there is no limit on the introduction of new field names, each presumably defining new semantics, or on the number of header fields used in a given message. Existing fields are defined in each part of this specification and in many other specifications outside the standards process. New header fields can be introduced without changing the protocol version if their defined semantics allow them to be safely ignored by recipients that do not recognize them.

New HTTP header fields SHOULD be registered with IANA according to the procedures in Section 3.1 of [Part2]. Unrecognized header fields MUST be forwarded by a proxy unless the field-name is listed in the Connection header field (Section 6.1) or the proxy is specifically configured to block or otherwise transform such fields. Unrecognized header fields SHOULD be ignored by other recipients.

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 ([RFC6265]). (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.

3.2.1. Whitespace

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 either be replaced with a single SP or transformed to all SP octets (each octet other than SP replaced with 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 either be replaced with a single SP or transformed to all SP octets 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          = *( SP / HTAB )
              ; "optional" whitespace
RWS          = 1*( SP / HTAB )
              ; "required" whitespace
BWS          = OWS
              ; "bad" whitespace
```

3.2.2. Field Parsing

No whitespace is allowed between the header field-name and colon. In the past, differences in the handling of such whitespace have led to security vulnerabilities in request routing and response handling. Any received request message that contains whitespace between a header field-name and colon **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).

Historically, HTTP header field values could be extended over multiple lines by preceding each extra line with at least one space or horizontal tab (obs-fold). This specification deprecates such line folding except within the message/http media type (Section 7.3.1). HTTP senders **MUST NOT** produce messages that include line folding (i.e., that contain any field-value that matches the obs-fold rule) unless the message is intended for packaging within the message/http media type. HTTP recipients **SHOULD** accept line folding and replace any embedded obs-fold whitespace with either a single SP or a matching number of SP octets (to avoid buffer copying) 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.

3.2.3. Field Length

HTTP does not place a pre-defined limit on the length of header fields, either in isolation or as a set. A server **MUST** be prepared to receive request header fields of unbounded length and respond with a 4xx (Client Error) status code if the received header field(s) would be longer than the server wishes to handle.

A client that receives response header fields that are longer than it wishes to handle can only treat it as a server error.

Various ad-hoc limitations on header field length are found in practice. It is RECOMMENDED that all HTTP senders and recipients support messages whose combined header fields have 4000 or more octets.

3.2.4. Field value components

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 4).

```

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
obs-text      = %x80-FF

```

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

```

quoted-pair    = "\" ( HTAB / SP / VCHAR / obs-text )

```

Recipients that process the value of the quoted-string MUST handle a quoted-pair as if it were replaced by the octet following the backslash.

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

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
```

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

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

Senders SHOULD NOT escape octets in comments 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 of that request or response. The message body is identical to the payload body unless a transfer coding has been applied, as described in Section 3.3.1.

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

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 a Content-Length or Transfer-Encoding header field. Request message framing is independent of method semantics, even if the method does not define any use for a message body.

The presence of a message body in a response depends on both the request method to which it is responding and the response status code (Section 3.1.2). 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. 2xx (Successful) responses to CONNECT switch to tunnel mode instead of having a message body. 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.

3.3.1. Transfer-Encoding

When one or more transfer codings are applied to a payload body in order to form the message body, a Transfer-Encoding header field MUST be sent in the message and MUST contain the list of corresponding transfer-coding names in the same order that they were applied. Transfer codings are defined in Section 4.

Transfer-Encoding = 1#transfer-coding

Transfer-Encoding is analogous to the Content-Transfer-Encoding field of MIME, which was 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's case, Transfer-Encoding is primarily intended to accurately delimit a dynamically generated payload and to distinguish payload encodings that are only applied for transport efficiency or security from those that are characteristics of the target resource.

The "chunked" transfer-coding (Section 4.1) MUST be implemented by all HTTP/1.1 recipients because it plays a crucial role in delimiting messages when the payload body size is not known in advance. When the "chunked" transfer-coding is used, it MUST be the last transfer-coding applied to form the message body and MUST NOT be applied more than once in a message body. If any transfer-coding is applied to a request payload body, the final transfer-coding applied MUST be "chunked". If any 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.

For example,

Transfer-Encoding: gzip, chunked

indicates that the payload body has been compressed using the gzip coding and then chunked using the chunked coding while forming the message body.

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.

Unlike Content-Encoding (Section 5.4 of [Part2]), 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. Additional information about the encoding parameters MAY be provided by other header fields not defined by this specification.

Transfer-Encoding MAY be sent in a response to a HEAD request or in a 304 (Not Modified) response (Section 4.1 of [Part4]) to a GET request, neither of which includes a message body, to indicate that the origin server would have applied a transfer coding to the message body if the request had been an unconditional GET. This indication is not required, however, because any recipient on the response chain (including the origin server) can remove transfer codings when they

are not needed.

Transfer-Encoding was added in HTTP/1.1. It is generally assumed that implementations advertising only HTTP/1.0 support will not understand how to process a transfer-encoded payload. A client **MUST NOT** send a request containing Transfer-Encoding unless it knows the server will handle HTTP/1.1 (or later) requests; such knowledge might be in the form of specific user configuration or by remembering the version of a prior received response. A server **MUST NOT** send a response containing Transfer-Encoding unless the corresponding request indicates HTTP/1.1 (or later).

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.

3.3.2. Content-Length

When a message does not have a Transfer-Encoding header field and the payload body length can be determined prior to being transferred, a Content-Length header field **SHOULD** be sent to indicate the length of the payload body that is either present as the message body, for requests and non-HEAD responses other than 304 (Not Modified), or would have been present had the request been an unconditional GET. The length is expressed as a decimal number of octets.

Content-Length = 1*DIGIT

An example is

Content-Length: 3495

In the case of a response to a HEAD request, Content-Length indicates the size of the payload body (without any potential transfer-coding) that would have been sent had the request been a GET. In the case of a 304 (Not Modified) response (Section 4.1 of [Part4]) to a GET request, Content-Length indicates the size of the payload body (without any potential transfer-coding) that would have been sent in a 200 (OK) response.

Any Content-Length field value greater than or equal to zero is valid. Since there is no predefined limit to the length of an HTTP payload, recipients **SHOULD** anticipate potentially large decimal numerals and prevent parsing errors due to integer conversion overflows (Section 8.6).

If a message is received that has multiple Content-Length header fields 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** either reject the message as invalid or replace the duplicated field-values with a single valid Content-Length field containing that decimal value prior to determining the message body length.

Note: HTTP's use of Content-Length for message framing differs significantly from the same field's use in MIME, where it is an optional field used only within the "message/external-body" media-type.

3.3.3. Message Body Length

The length of a 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 1xx (Informational), 204 (No Content), or 304 (Not Modified) status code 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. Any 2xx (Successful) response to a CONNECT request implies that the connection will become a tunnel immediately after the empty line that concludes the header fields. A client **MUST** ignore any Content-Length or Transfer-Encoding header fields received in such a message.
3. If a Transfer-Encoding header field is present and the "chunked" transfer-coding (Section 4.1) 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 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.

4. 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.
5. 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 a connection and the connection has been closed by the server.
6. 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).
7. 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.

3.4. Handling Incomplete Messages

Request messages that are prematurely terminated, possibly due to a canceled 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 response that terminates in the middle of the header block (before the empty line is received) cannot be assumed to convey the full semantics of the response and MUST be treated as an error.

A message body that uses the chunked transfer encoding is incomplete if the zero-sized chunk that terminates the encoding has not been received. A message that uses a valid Content-Length is incomplete if the size of the message body received (in octets) is less than the value given by Content-Length. A response that has neither chunked transfer encoding nor Content-Length is terminated by closure of the connection, and thus is considered complete regardless of the number of message body octets received, provided that the header block was received intact.

A user agent MUST NOT render an incomplete response message body as

if it were complete (i.e., some indication needs to be given to the user that an error occurred). Cache requirements for incomplete responses are defined in Section 3 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 6.3.2.2.

3.5. Message Parsing Robustness

Older HTTP/1.0 client implementations might 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.

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. Likewise, although the line terminator for the start-line and header fields is the sequence CRLF, we recommend that recipients recognize a single LF as a line terminator and ignore any CR.

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.

4. 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 4.1
                      / "compress" ; Section 4.2.1
                      / "deflate" ; Section 4.2.2
                      / "gzip" ; Section 4.2.3
                      / 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. The HTTP Transfer Coding registry is defined in Section 7.4. HTTP/1.1 uses transfer-coding values in the TE header field (Section 4.3) and in the Transfer-Encoding header field (Section 3.3.1).

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

chunk-ext          = *( ";" chunk-ext-name [ "=" chunk-ext-val ] )
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          = HTAB / SP / %x21 / %x23-5B / %x5D-7E / 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 4.4).

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 4.3; 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 field (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 conformance 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.

Use of chunk-ext extensions by senders is deprecated; they SHOULD NOT be sent and definition of new chunk-extensions is discouraged.

4.2. 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.

4.2.1. 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).

4.2.2. 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.

4.2.3. 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.

4.3. TE

The "TE" header field indicates what extension transfer-codings the client 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 4).

```
TE           = #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 4.1. 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 6.1) 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 4.3.1, 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 acceptable transfer-coding is "chunked". A message with no transfer-coding is always acceptable.

4.3.1. Quality Values

Both transfer codings (TE request header field, Section 4.3) and content negotiation (Section 8 of [Part2]) 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.

$$\text{qvalue} = \left(\frac{\text{"0" ["." 0*3DIGIT] }}{\text{"1" ["." 0*3("0")] }} \right)$$

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

4.4. 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 = 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 4.1 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:

- o Transfer-Encoding
- o Content-Length
- o Trailer

5. Message Routing

HTTP request message routing is determined by each client based on the target resource, the client's proxy configuration, and establishment or reuse of an inbound connection. The corresponding response routing follows the same connection chain back to the client.

5.1. Identifying a Target Resource

HTTP is used in a wide variety of applications, ranging from general-purpose computers to home appliances. In some cases, communication options are hard-coded in a client's configuration. However, most HTTP clients rely on the same resource identification mechanism and configuration techniques as general-purpose Web browsers.

HTTP communication is initiated by a user agent for some purpose. The purpose is a combination of request semantics, which are defined in [Part2], and a target resource upon which to apply those semantics. A URI reference (Section 2.8) is typically used as an identifier for the "target resource", which a user agent would resolve to its absolute form in order to obtain the "target URI". The target URI excludes the reference's fragment identifier component, if any, since fragment identifiers are reserved for client-side processing ([RFC3986], Section 3.5).

HTTP intermediaries obtain the request semantics and target URI from the request-line of an incoming request message.

5.2. Connecting Inbound

Once the target URI is determined, a client needs to decide whether a network request is necessary to accomplish the desired semantics and, if so, where that request is to be directed.

If the client has a response cache and the request semantics can be satisfied by a cache ([Part6]), then the request is usually directed to the cache first.

If the request is not satisfied by a cache, then a typical client will check its configuration to determine whether a proxy is to be used to satisfy the request. Proxy configuration is implementation-

dependent, but is often based on URI prefix matching, selective authority matching, or both, and the proxy itself is usually identified by an "http" or "https" URI. If a proxy is applicable, the client connects inbound by establishing (or reusing) a connection to that proxy.

If no proxy is applicable, a typical client will invoke a handler routine, usually specific to the target URI's scheme, to connect directly to an authority for the target resource. How that is accomplished is dependent on the target URI scheme and defined by its associated specification, similar to how this specification defines origin server access for resolution of the "http" (Section 2.8.1) and "https" (Section 2.8.2) schemes.

5.3. Request Target

Once an inbound connection is obtained (Section 6), the client sends an HTTP request message (Section 3) with a request-target derived from the target URI. There are four distinct formats for the request-target, depending on both the method being requested and whether the request is to a proxy.

```
request-target = origin-form
                / absolute-form
                / authority-form
                / asterisk-form

origin-form    = path-absolute [ "?" query ]
absolute-form  = absolute-URI
authority-form  = authority
asterisk-form   = "*"

```

The most common form of request-target is the origin-form. When making a request directly to an origin server, other than a CONNECT or server-wide OPTIONS request (as detailed below), a client MUST send only the absolute path and query components of the target URI as the request-target. If the target URI's path component is empty, then the client MUST send "/" as the path within the origin-form of request-target. A Host header field is also sent, as defined in Section 5.4, containing the target URI's authority component (excluding any userinfo).

For example, a client wishing to retrieve a representation of the resource identified as

```
http://www.example.org/where?q=now
```

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 /where?q=now HTTP/1.1
Host: www.example.org
```

followed by the remainder of the request message.

When making a request to a proxy, other than a CONNECT or server-wide OPTIONS request (as detailed below), a client MUST send the target URI in absolute-form as the request-target. The proxy is requested to either service that request from a valid cache, if possible, or make the same request on the client's behalf to either the next inbound proxy server or directly to the origin server indicated by the request-target. Requirements on such "forwarding" of messages are defined in Section 5.6.

An example absolute-form of request-line would be:

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

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

The authority-form of request-target is only used for CONNECT requests (Section 2.3.8 of [Part2]). When making a CONNECT request to establish a tunnel through one or more proxies, a client MUST send only the target URI's authority component (excluding any userinfo) as the request-target. For example,

```
CONNECT www.example.com:80 HTTP/1.1
```

The asterisk-form of request-target is only used for a server-wide OPTIONS request (Section 2.3.1 of [Part2]). When a client wishes to request OPTIONS for the server as a whole, as opposed to a specific named resource of that server, the client MUST send only "*" (%x2A) as the request-target. For example,

```
OPTIONS * HTTP/1.1
```

If a proxy receives an OPTIONS request with an absolute-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 send 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".

5.4. Host

The "Host" header field in a request provides the host and port information from the target URI, enabling the origin server to distinguish among 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 = uri-host [ ":" port ] ; Section 2.8.1
```

A client **MUST** send a Host header field in all HTTP/1.1 request messages. If the target URI includes an authority component, then the Host field-value **MUST** be identical to that authority component after excluding any userinfo (Section 2.8.1). If the authority component is missing or undefined for the target 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 absolute-form, 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 an absolute-form of request-target, the proxy **MUST** ignore the received Host header field (if any) and instead replace it with the host information of the request-target. If the proxy forwards the request, it **MUST** generate a new Host field-value based on the received request-target rather than forward the received Host field-value.

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 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.

5.5. Effective Request URI

A server that receives an HTTP request message **MUST** reconstruct the user agent's original target URI, based on the pieces of information learned from the request-target, Host header field, and connection context, in order to identify the intended target resource and properly service the request. The URI derived from this reconstruction process is referred to as the "effective request URI".

For a user agent, the effective request URI is the target URI.

If the request-target is in absolute-form, then the effective request URI is the same as the request-target. Otherwise, the effective request URI is constructed as follows.

If the request is received over an SSL/TLS-secured TCP connection, then the effective request URI's scheme is "https"; otherwise, the scheme is "http".

If the request-target is in authority-form, then the effective request URI's authority component is the same as the request-target. Otherwise, if a Host header field is supplied with a non-empty field-value, then the authority component is the same as the Host field-value. Otherwise, the authority component is the concatenation of the default host name configured for the server, a colon (":"), and the connection's incoming TCP port number in decimal form.

If the request-target is in authority-form or asterisk-form, then the effective request URI's combined path and query component is empty. Otherwise, the combined path and query component is the same as the request-target.

The components of the effective request URI, once determined as above, can be combined into absolute-URI form by concatenating the scheme, "://", authority, and combined path and query component.

Example 1: the following message received over an insecure TCP connection

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

has an effective request URI of

```
http://www.example.org:8080/pub/WWW/TheProject.html
```

Example 2: the following message received over an SSL/TLS-secured TCP connection

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

has an effective request URI of

```
https://www.example.org
```

An origin server that does not allow resources to differ by requested host MAY ignore the Host field-value and instead replace it with a configured server name when constructing the effective request URI.

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 guess the effective request URI's authority component.

5.6. Intermediary Forwarding

As described in Section 2.4, intermediaries can serve a variety of roles in the processing of HTTP requests and responses. Some intermediaries are used to improve performance or availability. Others are used for access control or to filter content. Since an HTTP stream has characteristics similar to a pipe-and-filter architecture, there are no inherent limits to the extent an intermediary can enhance (or interfere) with either direction of the stream.

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, or literal IP addresses.

If a proxy receives a request-target with a host name that is not a fully qualified domain name, it MAY add its domain to the host name it received when forwarding the request. A proxy MUST NOT change the

host name if it is a fully qualified domain name.

A non-transforming proxy MUST NOT rewrite the "path-absolute" and "query" parts of the received request-target when forwarding it to the next inbound server, except as noted above to replace an empty path with "/" or "*".

Intermediaries that forward a message MUST implement the Connection header field as specified in Section 6.1.

5.6.1. 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:

- o 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.
- o 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:

- o Connection
- o Keep-Alive (Section 19.7.1.1 of [RFC2068])
- o Proxy-Authenticate (Section 4.2 of [Part7])
- o Proxy-Authorization (Section 4.3 of [Part7])
- o TE
- o Trailer
- o Transfer-Encoding
- o 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 6.1).

5.6.2. 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:

- o Allow (Section 9.5 of [Part2])
- o Content-Location (Section 9.8 of [Part2])
- o Content-MD5 (Section 14.15 of [RFC2616])
- o ETag (Section 2.3 of [Part4])
- o Last-Modified (Section 2.2 of [Part4])
- o Server (Section 9.17 of [Part2])

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

- o Expires (Section 7.3 of [Part6])

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:

- o Content-Encoding (Section 9.6 of [Part2])
- o Content-Range (Section 5.2 of [Part5])
- o Content-Type (Section 9.9 of [Part2])

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 7.6 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 ([Part2]), though it MAY change the message body through application or removal of a transfer-coding (Section 4).

5.7. Associating a Response to a Request

HTTP does not include a request identifier for associating a given request message with its corresponding one or more response messages. Hence, it relies on the order of response arrival to correspond exactly to the order in which requests are made on the same connection. More than one response message per request only occurs when one or more informational responses (1xx, see Section 4.3 of [Part2]) precede a final response to the same request.

A client that uses persistent connections and sends more than one request per connection MUST maintain a list of outstanding requests in the order sent on that connection and MUST associate each received response message to the highest ordered request that has not yet received a final (non-1xx) response.

6. Connection Management

6.1. 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          = 1#connection-option
connection-option = token
```

Connection options are compared case-insensitively.

A proxy or gateway MUST parse a received Connection header field before a message is forwarded and, for each connection-option in this field, remove any header field(s) from the message with the same name as the connection-option, 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 7.2 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-option 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 conformant.

When defining new connection options, specifications ought to carefully consider existing deployed header fields and ensure that the new connection option does not share the same name as an unrelated header field that might already be deployed. Defining a new connection option 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 6.3) 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.

6.2. 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                = 1#( received-protocol RWS received-by
                        [ RWS comment ] )
received-protocol = [ protocol-name "/" ] protocol-version
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.

6.3. Persistent Connections

6.3.1. 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 [Niel1997]. Alternatives have also been explored, for example, T/TCP [Tou1998].

Persistent HTTP connections have a number of advantages:

- o 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.

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

6.3.2. 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 6.1). Once a close has been signaled, the client MUST NOT send any more requests on that connection.

6.3.2.1. 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 option "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 option "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 option "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 option "close".

If either the client or the server sends the "close" option 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 A.1.2 for more information on backward compatibility with HTTP/1.0 clients.

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).

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.

6.3.2.2. 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 2.1.2 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.

6.3.3. 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.

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.

6.3.4. Retrying Requests

Senders can close the transport connection at any time. Therefore, clients, servers, and proxies MUST be able to recover from asynchronous close events. Client software MAY reopen the transport

connection and retransmit the aborted sequence of requests without user interaction so long as the request sequence is idempotent (see Section 2.1.2 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.

6.4. Message Transmission Requirements

6.4.1. 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.

6.4.2. 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 4), 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.

6.4.3. Use of the 100 (Continue) Status

The purpose of the 100 (Continue) status code (see Section 4.3.1 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:

- o If a client will wait for a 100 (Continue) response before sending the request body, it MUST send an Expect header field (Section 9.11 of [Part2]) with the "100-continue" expectation.
- o A client MUST NOT send an Expect header field 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:

- o 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.
- o 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.
- o 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.
- o 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.
- o 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 ought 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:

- o 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.
- o 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.
- o Proxies SHOULD maintain a record of the HTTP version numbers received from recently-referenced next-hop servers.
- o 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 lxx responses (see Section 4.3 of [Part2]).

6.4.4. Closing Connections on Error

If the client is sending data, a server implementation using TCP SHOULD be careful to ensure that the client acknowledges receipt of the packet(s) containing the response, before the server closes the input connection. If the client continues sending data to the server after the close, the server's TCP stack will send a reset packet to the client, which might erase the client's unacknowledged input buffers before they can be read and interpreted by the HTTP application.

6.5. 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           = 1#protocol

protocol          = protocol-name [ "/" protocol-version ]
protocol-name     = token
protocol-version  = token
```

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 transitioning 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 6.1) 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 4.5 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 2.7 and future updates to this specification. Additional tokens can be registered with IANA using the registration procedure defined in Section 7.6.

7. IANA Considerations

7.1. Header Field Registration

HTTP header fields are registered within the Message Header Field Registry [RFC3864] maintained by IANA at <http://www.iana.org/assignments/message-headers/message-header-index.html>.

This document defines the following HTTP header fields, so their associated registry entries shall be updated according to the permanent registrations below:

Header Field Name	Protocol	Status	Reference
Connection	http	standard	Section 6.1
Content-Length	http	standard	Section 3.3.2
Host	http	standard	Section 5.4
TE	http	standard	Section 4.3
Trailer	http	standard	Section 4.4
Transfer-Encoding	http	standard	Section 3.3.1
Upgrade	http	standard	Section 6.5
Via	http	standard	Section 6.2

Furthermore, the header field-name "Close" shall be registered as "reserved", since using that name as an HTTP header field might conflict with the "close" connection option of the "Connection" header field (Section 6.1).

Header Field Name	Protocol	Status	Reference
Close	http	reserved	Section 7.1

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

7.2. URI Scheme Registration

IANA maintains the registry of URI Schemes [RFC4395] at
<<http://www.iana.org/assignments/uri-schemes.html>>.

This document defines the following URI schemes, so their associated registry entries shall be updated according to the permanent registrations below:

URI Scheme	Description	Reference
http	Hypertext Transfer Protocol	Section 2.8.1
https	Hypertext Transfer Protocol Secure	Section 2.8.2

7.3. 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]).

7.3.1. 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 7.3.1).

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

7.3.2. 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 7.3.2).

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

7.4. Transfer Coding Registry

The HTTP Transfer Coding Registry defines the name space for transfer coding names.

Registrations MUST include the following fields:

- o Name
- o Description
- o Pointer to specification text

Names of transfer codings MUST NOT overlap with names of content codings (Section 5.4 of [Part2]) unless the encoding transformation

is identical, as it is the case for the compression codings defined in Section 4.2.

Values to be added to this name space require IETF Review (see 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>>.

7.5. Transfer Coding Registrations

The HTTP Transfer Coding Registry shall be updated with the registrations below:

Name	Description	Reference
chunked	Transfer in a series of chunks	Section 4.1
compress	UNIX "compress" program method	Section 4.2.1
deflate	"deflate" compression mechanism ([RFC1951]) used inside the "zlib" data format ([RFC1950])	Section 4.2.2
gzip	Same as GNU zip [RFC1952]	Section 4.2.3

7.6. Upgrade Token Registry

The HTTP Upgrade Token Registry defines the name space for protocol-name tokens used to identify protocols in the Upgrade header field. Each registered protocol name 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 happen on a "First Come First Served" basis (see Section 4.1 of [RFC5226]) and are subject to the following rules:

1. A protocol-name 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 registration SHOULD name a set of expected "protocol-version" tokens associated with that token at the time of registration.
6. 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.
7. The IESG MAY reassign responsibility for a protocol token. This will normally only be used in the case when a responsible party cannot be contacted.

This registration procedure for HTTP Upgrade Tokens replaces that previously defined in Section 7.2 of [RFC2817].

7.7. Upgrade Token Registration

The HTTP Upgrade Token Registry shall be updated with the registration below:

Value	Description	Expected Version Tokens	Reference
HTTP	Hypertext Transfer Protocol	any DIGIT.DIGIT (e.g, "2.0")	Section 2.7

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

8. Security Considerations

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

8.1. 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 implementers 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 implementer's company.

8.2. 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. In particular, log information gathered at an intermediary often contains a history of user agent interaction, across a multitude of sites, that can be traced to individual users.

HTTP log information is confidential in nature; its handling is often constrained by laws and regulations. Log information needs to be securely stored and appropriate guidelines followed for its analysis. Anonymization of personal information within individual entries helps, but is generally not sufficient to prevent real log traces from being re-identified based on correlation with other access characteristics. As such, access traces that are keyed to a specific client should not be published even if the key is pseudonymous.

To minimize the risk of theft or accidental publication, log information should be purged of personally identifiable information, including user identifiers, IP addresses, and user-provided query parameters, as soon as that information is no longer necessary to support operational needs for security, auditing, or fraud control.

8.3. 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.

8.4. DNS-related Attacks

HTTP clients rely heavily on the Domain Name Service (DNS), and are thus generally prone to security attacks based on the deliberate misassociation of IP addresses and DNS names not protected by DNSSec. Clients need to be cautious in assuming the validity of an IP number/DNS name association unless the response is protected by DNSSec ([RFC4033]).

8.5. Intermediaries and Caching

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

Intermediaries that contain a shared cache are especially vulnerable to cache poisoning attacks.

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

Users need to be aware that intermediaries are no more trustworthy 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.

8.6. Protocol Element Size Overflows

Because HTTP uses mostly textual, character-delimited fields, attackers can overflow buffers in implementations, and/or perform a Denial of Service against implementations that accept fields with unlimited lengths.

To promote interoperability, this specification makes specific recommendations for minimum size limits on request-line (Section 3.1.1) and blocks of header fields (Section 3.2). These are minimum recommendations, chosen to be supportable even by

implementations with limited resources; it is expected that most implementations will choose substantially higher limits.

This specification also provides a way for servers to reject messages that have request-targets that are too long (Section 4.6.12 of [Part2]) or request entities that are too large (Section 4.6 of [Part2]).

Other fields (including but not limited to request methods, response status phrases, header field-names, and body chunks) SHOULD be limited by implementations carefully, so as to not impede interoperability.

9. Acknowledgments

This edition of HTTP builds on the many contributions that went into RFC 1945, RFC 2068, RFC 2145, and RFC 2616, including substantial contributions made by the previous authors, editors, and working group chairs: Tim Berners-Lee, Ari Luotonen, Roy T. Fielding, Henrik Frystyk Nielsen, Jim Gettys, Jeffrey C. Mogul, Larry Masinter, Paul J. Leach, and Mark Nottingham. See Section 16 of [RFC2616] for additional acknowledgements from prior revisions.

Since 1999, the following contributors have helped improve the HTTP specification by reporting bugs, asking smart questions, drafting or reviewing text, and evaluating open issues:

Adam Barth, Adam Roach, Addison Phillips, Adrian Chadd, Adrien W. de Croy, Alan Ford, Alan Ruttenberg, Albert Lunde, Alek Storm, Alex Rousskov, Alexandre Morgaut, Alexey Melnikov, Alisha Smith, Amichai Rothman, Amit Klein, Amos Jeffries, Andreas Maier, Andreas Petersson, Anne van Kesteren, Anthony Bryan, Asbjorn Ulsberg, Balachander Krishnamurthy, Barry Leiba, Ben Laurie, Benjamin Niven-Jenkins, Bill Corry, Bill Burke, Bjoern Hoehrmann, Bob Scheifler, Boris Zbarsky, Brett Slatkin, Brian Kell, Brian McBarron, Brian Pane, Brian Smith, Bryce Nesbitt, Cameron Heaven-Jones, Carl Kugler, Carsten Bormann, Charles Fry, Chris Newman, Cyrus Daboo, Dale Robert Anderson, Dan Winship, Daniel Stenberg, Dave Cridland, Dave Crocker, Dave Kristol, David Booth, David Singer, David W. Morris, Diwakar Shetty, Dmitry Kurochkin, Drummond Reed, Duane Wessels, Edward Lee, Eliot Lear, Eran Hammer-Lahav, Eric D. Williams, Eric J. Bowman, Eric Lawrence, Eric Rescorla, Erik Aronesty, Florian Weimer, Frank Ellermann, Fred Bohle, Geoffrey Sneddon, Gervase Markham, Greg Wilkins, Harald Tveit Alvestrand, Harry Halpin, Helge Hess, Henrik Nordstrom, Henry S. Thompson, Henry Story, Herbert van de Sompel, Howard Melman, Hugo Haas, Ian Hickson, Ingo Struck, J. Ross Nicoll, James H. Manger, James Lacey, James M. Snell, Jamie Lokier, Jan Algermissen, Jeff Hodges (who came up with the term 'effective Request-URI'), Jeff

Walden, Jim Luther, Joe D. Williams, Joe Gregorio, Joe Orton, John C. Klensin, John C. Mallery, John Cowan, John Kemp, John Panzer, John Schneider, John Stracke, John Sullivan, Jonas Sicking, Jonathan Billington, Jonathan Moore, Jonathan Rees, Jonathan Silvera, Jordi Ros, Joris Dobbelsteen, Josh Cohen, Julien Pierre, Jungshik Shin, Justin Chapweske, Justin Erenkrantz, Justin James, Kalvinder Singh, Karl Dubost, Keith Hoffman, Keith Moore, Koen Holtman, Konstantin Voronkov, Kris Zyp, Lisa Dusseault, Maciej Stachowiak, Marc Schneider, Marc Slemko, Mark Baker, Mark Pauley, Mark Watson, Markus Isomaki, Markus Lanthaler, Martin J. Duerst, Martin Musatov, Martin Nilsson, Martin Thomson, Matt Lynch, Matthew Cox, Max Clark, Michael Burrows, Michael Hausenblas, Mike Amundsen, Mike Belshe, Mike Kelly, Mike Schinkel, Miles Sabin, Murray S. Kucherawy, Mykyta Yevstifeyev, Nathan Rixham, Nicholas Shanks, Nico Williams, Nicolas Alvarez, Nicolas Mailhot, Noah Slater, Pablo Castro, Pat Hayes, Patrick R. McManus, Paul E. Jones, Paul Hoffman, Paul Marquess, Peter Lepeska, Peter Saint-Andre, Peter Watkins, Phil Archer, Phillip Hallam-Baker, Poul-Henning Kamp, Preethi Natarajan, Ray Polk, Reto Bachmann-Gmuer, Richard Cyganiak, Robert Brewer, Robert Collins, Robert O'Callahan, Robert Olofsson, Robert Sayre, Robert Siemer, Robert de Wilde, Roberto Javier Godoy, Roberto Peon, Ronny Widjaja, S. Mike Dierken, Salvatore Loreto, Sam Johnston, Sam Ruby, Scott Lawrence (who maintained the original issues list), Sean B. Palmer, Shane McCarron, Stefan Eissing, Stefan Tilkov, Stefanos Harhalakis, Stephane Bortzmeyer, Stephen Farrell, Stephen Ludin, Stuart Williams, Subbu Allamaraju, Sylvain Hellegouarch, Tapan Divekar, Tatsuya Hayashi, Ted Hardie, Thomas Broyer, Thomas Nordin, Thomas Roessler, Tim Bray, Tim Morgan, Tim Olsen, Tom Zhou, Travis Snoozy, Tyler Close, Vincent Murphy, Wenbo Zhu, Werner Baumann, Wilbur Streett, Wilfredo Sanchez Vega, William A. Rowe Jr., William Chan, Willy Tarreau, Xiaoshu Wang, Yaron Goland, Yngve Nysaeter Pettersen, Yoav Nir, Yogesh Bang, Yutaka Oiwa, Zed A. Shaw, and Zhong Yu.

10. References

10.1. Normative References

- [Part2] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 2: Semantics and Payloads", draft-ietf-httpbis-p2-semantics-20 (work in progress), July 2012.
- [Part4] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 4: Conditional Requests", draft-ietf-httpbis-p4-conditional-20 (work in progress), July 2012.
- [Part5] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed.,

"HTTP/1.1, part 5: Range Requests",
draft-ietf-httpbis-p5-range-20 (work in progress),
July 2012.

[Part6] Fielding, R., Ed., Lafon, Y., Ed., Nottingham, M., Ed.,
and J. Reschke, Ed., "HTTP/1.1, part 6: Caching",
draft-ietf-httpbis-p6-cache-20 (work in progress),
July 2012.

[Part7] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed.,
"HTTP/1.1, part 7: Authentication",
draft-ietf-httpbis-p7-auth-20 (work in progress),
July 2012.

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

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

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

[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.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for
Syntax Specifications: ABNF", STD 68, RFC 5234,
January 2008.

[USASCII] American National Standards Institute, "Coded Character
Set -- 7-bit American Standard Code for Information
Interchange", ANSI X3.4, 1986.

10.2. Informative 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.

[Kri2001] Kristol, D., "HTTP Cookies: Standards, Privacy, and
Politics", ACM Transactions on Internet Technology Vol.

- 1, #2, November 2001,
<<http://arxiv.org/abs/cs.SE/0105018>>.
- [Niel997] 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,
<<http://doi.acm.org/10.1145/263105.263157>>.
- [Pad1995] Padmanabhan, V. and J. Mogul, "Improving HTTP Latency", Computer Networks and ISDN Systems v. 28, pp. 25-35, December 1995,
<<http://portal.acm.org/citation.cfm?id=219094>>.
- [RFC1919] Chatel, M., "Classical versus Transparent IP Proxies", RFC 1919, March 1996.
- [RFC1945] Berners-Lee, T., Fielding, R., and H. Nielsen, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.
- [RFC2045] Freed, N. and N. 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., Fielding, R., Gettys, J., and H. 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. 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.
- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.
- [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.
- [RFC4559] Jaganathan, K., Zhu, L., and J. Brezak, "SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows", RFC 4559, June 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.
- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.
- [Spe] Spero, S., "Analysis of HTTP Performance Problems", <<http://sunsite.unc.edu/mdma-release/http-prob.html>>.
- [Tou1998] Touch, J., Heidemann, J., and K. Obraczka, "Analysis of HTTP Performance", ISI Research Report ISI/RR-98-463, Aug 1998, <<http://www.isi.edu/touch/pubs/http-perf96/>>.
- (original report dated Aug. 1996)

Appendix A. 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 conformance with HTTP/1.1.

It is beyond the scope of a protocol specification to mandate conformance 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, we 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.

A.1. Changes from HTTP/1.0

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

A.1.1.1. Multi-homed Web Servers

The requirements that clients and servers support the Host header field (Section 5.4), report an error if it is missing from an HTTP/1.1 request, and accept absolute URIs (Section 5.3) 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.

A.1.1.2. Keep-Alive Connections

In 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 explicitly negotiated ("Keep-Alive") version of persistent connections described in Section 19.7.1 of [RFC2068].

Some clients and servers might wish to be compatible with these previous approaches to persistent connections, by explicitly negotiating for them with a "Connection: keep-alive" request header field. However, some experimental implementations of HTTP/1.0 persistent connections are faulty; for example, if a HTTP/1.0 proxy server doesn't understand Connection, it will erroneously forward that header field to the next inbound server, which would result in a hung connection.

One attempted solution was the introduction of a Proxy-Connection header field, targeted specifically at proxies. In practice, this was also unworkable, because proxies are often deployed in multiple layers, bringing about the same problem discussed above.

As a result, clients are encouraged not to send the Proxy-Connection header field in any requests.

Clients are also encouraged to consider the use of Connection: keep-alive in requests carefully; while they can enable persistent connections with HTTP/1.0 servers, clients using them need will need to monitor the connection for "hung" requests (which indicate that the client ought stop sending the header field), and this mechanism

ought not be used by clients at all when a proxy is being used.

A.1.3. Introduction of Transfer-Encoding

HTTP/1.1 introduces the Transfer-Encoding header field (Section 3.3.1). Proxies/gateways **MUST** remove any transfer-coding prior to forwarding a message via a MIME-compliant protocol.

A.2. Changes from RFC 2616

Clarify that the string "HTTP" in the HTTP-version ABNF production is case sensitive. Restrict the version numbers to be single digits due to the fact that implementations are known to handle multi-digit version numbers incorrectly. (Section 2.7)

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 5.3)

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

Rules about implicit linear whitespace between certain grammar productions have been removed; now whitespace is only allowed where specifically defined in the ABNF. (Section 3.2.1)

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 3.2.4)

Empty list elements in list productions have been deprecated. (Appendix B)

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 4)

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, and deprecate their use. (Section 4.1)

Registration of Transfer Codings now requires IETF Review (Section 7.4)

Remove hard limit of two connections per server. Remove requirement to retry a sequence of requests as long it was idempotent. Remove requirements about when servers are allowed to close connections prematurely. (Section 6.3.3)

Remove requirement to retry requests under certain circumstances when the server prematurely closes the connection. (Section 6.4)

Change ABNF productions for header fields to only define the field value.

Clarify exactly when "close" connection options have to be sent. (Section 6.1)

Define the semantics of the Upgrade header field in responses other than 101 (this was incorporated from [RFC2817]). (Section 6.5)

Take over the Upgrade Token Registry, previously defined in Section 7.2 of [RFC2817]. (Section 7.6)

Appendix B. ABNF list extension: #rule

A #rule extension to the ABNF rules of [RFC5234] is used to improve readability in the definitions of some header field values.

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).

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 3.2.4
```

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  "
```

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

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

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

Appendix C. Collected ABNF

```
BWS = OWS
```

```
Connection = *( "," OWS ) connection-option *( OWS "," [ OWS
  connection-option ] )
Content-Length = 1*DIGIT
```

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

```
HTTP-name = %x48.54.54.50 ; HTTP
```

```
HTTP-version = HTTP-name "/" DIGIT "." DIGIT
```

```
Host = uri-host [ ":" port ]
```

```
OWS = *( SP / HTAB )
```

```
RWS = 1*( SP / HTAB )
```

```
TE = [ ( "," / t-codings ) *( OWS "," [ OWS t-codings ] ) ]
```

```
Trailer = *( "," OWS ) field-name *( OWS "," [ OWS field-name ] )
```

```

Transfer-Encoding = *( "," OWS ) transfer-coding *( OWS "," [ OWS
  transfer-coding ] )

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

Via = *( "," 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>
absolute-form = absolute-URI
asterisk-form = "*"
attribute = token
authority = <authority, defined in [RFC3986], Section 3.2>
authority-form = authority

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

field-content = *( HTAB / SP / VCHAR / obs-text )
field-name = token
field-value = *( field-content / obs-fold )

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

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

message-body = *OCTET
method = token

obs-fold = CRLF ( SP / HTAB )
obs-text = %x80-FF
origin-form = path-absolute [ "?" query ]

```

```

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>
protocol = protocol-name [ "/" protocol-version ]
protocol-name = token
protocol-version = token
pseudonym = token

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

reason-phrase = *( HTAB / SP / VCHAR / obs-text )
received-by = ( uri-host [ ":" port ] ) / pseudonym
received-protocol = [ protocol-name "/" ] protocol-version
relative-part = <relative-part, defined in [RFC3986], Section 4.2>
request-line = method SP request-target SP HTTP-version CRLF
request-target = origin-form / absolute-form / authority-form /
  asterisk-form

special = "(" / ")" / "<" / ">" / "@" / "," / ";" / ":" / "\" /
  DQUOTE / "/" / "[" / "]" / "?" / "=" / "{" / "}"
start-line = request-line / status-line
status-code = 3DIGIT
status-line = HTTP-version SP status-code SP reason-phrase CRLF

t-codings = "trailers" / ( transfer-extension [ te-params ] )
tchar = "!" / "#" / "$" / "%" / "&" / "'" / "*" / "+" / "-" / "." /
  "^" / "_" / "`" / "|" / "~" / DIGIT / ALPHA
te-ext = OWS ";" OWS token [ "=" word ]
te-params = OWS ";" OWS "q=" qvalue *te-ext
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

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

D.1. Since RFC 2616

Extracted relevant partitions from [RFC2616].

D.2. Since draft-ietf-httpbis-pl-messaging-00

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/1>>: "HTTP Version should be case sensitive" (<http://purl.org/NET/http-errata#verscase>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/2>>: "'unsafe' characters" (<http://purl.org/NET/http-errata#unsafe-uri>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/3>>: "Chunk Size Definition" (<http://purl.org/NET/http-errata#chunk-size>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/4>>: "Message Length" (<http://purl.org/NET/http-errata#msg-len-chars>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/8>>: "Media Type Registrations" (<http://purl.org/NET/http-errata#media-reg>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/11>>: "URI includes query" (<http://purl.org/NET/http-errata#uriquery>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/15>>: "No close on lxx responses" (<http://purl.org/NET/http-errata#nocloselxx>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/16>>: "Remove 'identity' token references" (<http://purl.org/NET/http-errata#identity>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/26>>: "Import query BNF"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/31>>: "qdtex BNF"

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/35>>: "Normative and Informative references"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/42>>: "RFC2606 Compliance"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/45>>: "RFC977 reference"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/46>>: "RFC1700 references"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/47>>: "inconsistency in date format explanation"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/48>>: "Date reference typo"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/65>>: "Informative references"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/66>>: "ISO-8859-1 Reference"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/86>>: "Normative up-to-date references"

Other changes:

- o Update media type registrations to use RFC4288 template.
- o Use names of RFC4234 core rules DQUOTE and HTAB, fix broken ABNF for chunk-data (work in progress on <<http://tools.ietf.org/wg/httpbis/trac/ticket/36>>)

D.3. Since draft-ietf-httpbis-pl-messaging-01

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/19>>: "Bodies on GET (and other) requests"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/55>>: "Updating to RFC4288"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/57>>: "Status Code and Reason Phrase"

- o `<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>`):

- o Get rid of duplicate BNF rule names ("host" -> "uri-host", "trailer" -> "trailer-part").
- o Avoid underscore character in rule names ("http_URL" -> "http-URL", "abs_path" -> "path-absolute").
- o 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.
- o Synchronize core rules with RFC5234.
- o Get rid of prose rules that span multiple lines.
- o Get rid of unused rules LOALPHA and UPALPHA.
- o Move "Product Tokens" section (back) into Part 1, as "token" is used in the definition of the Upgrade header field.
- o Add explicit references to BNF syntax and rules imported from other parts of the specification.
- o Rewrite prose rule "token" in terms of "tchar", rewrite prose rule "TEXT".

D.4. Since draft-ietf-httpbis-pl-messaging-02

Closed issues:

- o `<http://tools.ietf.org/wg/httpbis/trac/ticket/51>`: "HTTP-date vs. rfc1123-date"
- o `<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>`):

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

Ongoing work on ABNF conversion

(<<http://tools.ietf.org/wg/httpbis/trac/ticket/36>>):

- o Replace string literals when the string really is case-sensitive (HTTP-version).

D.5. Since draft-ietf-httpbis-pl-messaging-03

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/28>>: "Connection closing"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/97>>: "Move registrations and registry information to IANA Considerations"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/120>>: "need new URL for PAD1995 reference"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/127>>: "IANA Considerations: update HTTP URI scheme registration"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/128>>: "Cite HTTPS URI scheme definition"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/129>>: "List-type header fields vs Set-Cookie"

Ongoing work on ABNF conversion

(<<http://tools.ietf.org/wg/httpbis/trac/ticket/36>>):

- o Replace string literals when the string really is case-sensitive (HTTP-Date).
- o Replace HEX by HEXDIG for future consistence with RFC 5234's core rules.

D.6. Since draft-ietf-httpbis-pl-messaging-04

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/34>>: "Out-of-date reference for URIs"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/132>>: "RFC 2822 is updated by RFC 5322"

Ongoing work on ABNF conversion

(<http://tools.ietf.org/wg/httpbis/trac/ticket/36>):

- o Use "/" instead of "|" for alternatives.
- o Get rid of RFC822 dependency; use RFC5234 plus extensions instead.
- o Only reference RFC 5234's core rules.
- o Introduce new ABNF rules for "bad" whitespace ("BWS"), optional whitespace ("OWS") and required whitespace ("RWS").
- o Rewrite ABNFs to spell out whitespace rules, factor out header field value format definitions.

D.7. Since draft-ietf-httpbis-pl-messaging-05

Closed issues:

- o <http://tools.ietf.org/wg/httpbis/trac/ticket/30>: "Header LWS"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/52>: "Sort 1.3 Terminology"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/63>: "RFC2047 encoded words"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/74>: "Character Encodings in TEXT"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/77>: "Line Folding"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/83>: "OPTIONS * and proxies"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/94>: "reason-phrase BNF"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/111>: "Use of TEXT"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/118>: "Join "Differences Between HTTP Entities and RFC 2045 Entities"?"
- o <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>):

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

Other changes:

- o Rewrite introduction; add mostly new Architecture Section.
- o 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).

D.8. Since draft-ietf-httpbis-pl-messaging-06

Closed issues:

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

Partly resolved issues:

- o <<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)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/163>>: "editorial improvements around HTTP-date"

D.9. Since draft-ietf-httpbis-pl-messaging-07

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/93>>: "Repeating single-value header fields"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/131>>: "increase connection limit"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/157>>: "IP addresses in URLs"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/172>>: "take over HTTP Upgrade Token Registry"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/173>>: "CR and LF in chunk extension values"

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/184>>: "HTTP/0.9 support"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/188>>: "pick IANA policy (RFC5226) for Transfer Coding / Content Coding"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/189>>: "move definitions of gzip/deflate/compress to part 1"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/194>>: "disallow control characters in quoted-pair"

Partly resolved issues:

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

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

Closed issues:

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

Partly resolved issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/60>>: "Placement of 13.5.1 and 13.5.2"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/200>>: "use of term "word" when talking about header field structure"

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

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/73>>: "Clarification of the term 'deflate'"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/83>>: "OPTIONS * and proxies"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/122>>: "MIME-Version not listed in P1, general header fields"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/143>>: "IANA registry for content/transfer encodings"

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/165>>: "Case-sensitivity of HTTP-date"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/200>>: "use of term "word" when talking about header field structure"

Partly resolved issues:

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

D.12. Since draft-ietf-httpbis-pl-messaging-10

Closed issues:

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

Partly resolved issues:

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

D.13. Since draft-ietf-httpbis-pl-messaging-11

Closed issues:

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

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/248>>: "confusing Date requirements for clients"

Partly resolved issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/95>>: "Handling multiple Content-Length header fields"

D.14. Since draft-ietf-httpbis-pl-messaging-12

Closed issues:

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

D.15. Since draft-ietf-httpbis-pl-messaging-13

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/53>>: "Allow is not in 13.5.2"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/95>>: "Handling multiple Content-Length header fields"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/276>>: "untangle ABNFs for header fields"

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/286>>: "Content-Length ABNF broken"

D.16. Since draft-ietf-httpbis-pl-messaging-14

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/273>>: "HTTP-version should be redefined as fixed length pair of DIGIT . DIGIT"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/282>>: "Recommend minimum sizes for protocol elements"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/283>>: "Set expectations around buffering"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/288>>: "Considering messages in isolation"

D.17. Since draft-ietf-httpbis-pl-messaging-15

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/100>>: "DNS Spoofing / DNS Binding advice"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/254>>: "move RFCs 2145, 2616, 2817 to Historic status"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/270>>: "\"-escaping in quoted strings"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/305>>: "'Close' should be reserved in the HTTP header field registry"

D.18. Since draft-ietf-httpbis-pl-messaging-16

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/186>>: "Document HTTP's error-handling philosophy"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/215>>: "Explain header field registration"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/219>>: "Revise Acknowledgements Sections"

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/297>>: "Retrying Requests"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/318>>: "Closing the connection on server error"

D.19. Since draft-ietf-httpbis-pl-messaging-17

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/158>>: "Proxy-Connection and Keep-Alive"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/166>>: "Clarify 'User Agent' "
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/300>>: "Define non-final responses"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/323>>: "intended maturity level vs normative references"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/324>>: "Intermediary rewriting of queries"

D.20. Since draft-ietf-httpbis-pl-messaging-18

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/250>>: "message-body in CONNECT response"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/302>>: "Misplaced text on connection handling in p2"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/335>>: "wording of line folding rule"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/343>>: "chunk-extensions"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/346>>: "make IANA policy definitions consistent"

D.21. Since draft-ietf-httpbis-pl-messaging-19

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/346>>: "make IANA policy definitions consistent"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/359>>: "clarify connection header field values are case-insensitive"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/361>>: "ABNF requirements for recipients"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/368>>: "note introduction of new IANA registries as normative changes"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/374>>: "Reference to ISO-8859-1 is informative"

Index

- A
 - absolute-form (of request-target) 41
 - accelerator 11
 - application/http Media Type 60
 - asterisk-form (of request-target) 41
 - authority-form (of request-target) 41
- B
 - browser 8
- C
 - cache 13
 - cacheable 13
 - captive portal 12
 - chunked (Coding Format) 34
 - client 7
 - Coding Format
 - chunked 34
 - compress 36
 - deflate 36
 - gzip 36
 - compress (Coding Format) 36
 - connection 7
 - Connection header field 47
 - Content-Length header field 29
- D

deflate (Coding Format)	36
downstream	11
E	
effective request URI	43
G	
gateway	11
Grammar	
absolute-form	40
absolute-URI	17
ALPHA	7
asterisk-form	40
attribute	34
authority	17
authority-form	40
BWS	24
chunk	34
chunk-data	34
chunk-ext	34
chunk-ext-name	34
chunk-ext-val	34
chunk-size	34
chunked-body	34
comment	27
Connection	47
connection-option	47
Content-Length	29
CR	7
CRLF	7
ctext	27
CTL	7
date2	34
date3	34
DIGIT	7
DQUOTE	7
field-content	23
field-name	23
field-value	23
header-field	23
HEXDIG	7
Host	42
HTAB	7
HTTP-message	20
HTTP-name	14
http-URI	17
HTTP-version	14
https-URI	19

last-chunk 34
LF 7
message-body 27
method 21
obs-fold 23
obs-text 26
OCTET 7
origin-form 40
OWS 24
partial-URI 17
path-absolute 17
port 17
protocol-name 49
protocol-version 49
pseudonym 49
qdtex 26
qdtex-nf 34
query 17
quoted-cpair 27
quoted-pair 26
quoted-str-nf 34
quoted-string 26
qvalue 38
reason-phrase 22
received-by 49
received-protocol 49
request-line 21
request-target 40
RWS 24
SP 7
special 26
start-line 21
status-code 22
status-line 22
t-codings 37
tchar 26
TE 37
te-ext 37
te-params 37
token 26
Trailer 38
trailer-part 34
transfer-coding 34
Transfer-Encoding 28
transfer-extension 34
transfer-parameter 34
Upgrade 57
uri-host 17

- URI-reference 17
- value 34
- VCHAR 7
- Via 49
- word 26
- gzip (Coding Format) 36

H

- header field 20
- Header Fields
 - Connection 47
 - Content-Length 29
 - Host 42
 - TE 36
 - Trailer 38
 - Transfer-Encoding 27
 - Upgrade 56
 - Via 49
- header section 20
- headers 20
- Host header field 42
- http URI scheme 17
- https URI scheme 18

I

- inbound 11
- interception proxy 12
- intermediary 10

M

- Media Type
 - application/http 60
 - message/http 59
- message 8
- message/http Media Type 59
- method 21

N

- non-transforming proxy 11

O

- origin server 8
- origin-form (of request-target) 40
- outbound 11

P

- proxy 11

R

recipient 8
request 8
request-target 21
resource 16
response 8
reverse proxy 11

S

sender 8
server 7
spider 8

T

target resource 39
target URI 39
TE header field 36
Trailer header field 38
Transfer-Encoding header field 27
transforming proxy 11
transparent proxy 12
tunnel 12

U

Upgrade header field 56
upstream 11
URI scheme
 http 17
 https 18
user agent 8

V

Via header field 49

Authors' Addresses

Roy T. Fielding (editor)
Adobe Systems Incorporated
345 Park Ave
San Jose, CA 95110
USA

EMail: fielding@gbiv.com
URI: <http://roy.gbiv.com/>

Yves Lafon (editor)
World Wide Web Consortium
W3C / ERCIM
2004, rte des Lucioles
Sophia-Antipolis, AM 06902
France

EMail: ylafon@w3.org
URI: <http://www.raubacapeu.net/people/yves/>

Julian F. Reschke (editor)
greenbytes GmbH
Hafenweg 16
Muenster, NW 48155
Germany

EMail: julian.reschke@greenbytes.de
URI: <http://greenbytes.de/tech/webdav/>

HTTPbis Working Group
Internet-Draft
Obsoletes: 2616 (if approved)
Updates: 2817 (if approved)
Intended status: Standards Track
Expires: January 17, 2013

R. Fielding, Ed.
Adobe
Y. Lafon, Ed.
W3C
J. Reschke, Ed.
greenbytes
July 16, 2012

HTTP/1.1, part 2: Semantics and Payloads
draft-ietf-httpbis-p2-semantics-20

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypertext information systems. This document defines the semantics of HTTP/1.1 messages, as expressed by request methods, request header fields, response status codes, and response header fields, along with the payload of messages (metadata and body content) and mechanisms for content negotiation.

Editorial Note (To be removed by RFC Editor)

Discussion of this draft takes place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org), which is archived at <http://lists.w3.org/Archives/Public/ietf-http-wg/>.

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 F.40.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference

material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 17, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	7
1.1. Conformance and Error Handling	7
1.2. Syntax Notation	8
2. Methods	8
2.1. Safe and Idempotent Methods	9
2.1.1. Safe Methods	9
2.1.2. Idempotent Methods	9
2.2. Method Registry	9
2.2.1. Considerations for New Methods	10
2.3. Method Definitions	10
2.3.1. OPTIONS	11
2.3.2. GET	12
2.3.3. HEAD	12
2.3.4. POST	13
2.3.5. PUT	14

2.3.6.	DELETE	16
2.3.7.	TRACE	16
2.3.8.	CONNECT	17
3.	Header Fields	18
3.1.	Considerations for Creating Header Fields	18
3.2.	Request Header Fields	20
3.3.	Response Header Fields	21
4.	Status Codes	22
4.1.	Overview of Status Codes	22
4.2.	Status Code Registry	24
4.2.1.	Considerations for New Status Codes	24
4.3.	Informational 1xx	25
4.3.1.	100 Continue	25
4.3.2.	101 Switching Protocols	25
4.4.	Successful 2xx	26
4.4.1.	200 OK	26
4.4.2.	201 Created	26
4.4.3.	202 Accepted	27
4.4.4.	203 Non-Authoritative Information	27
4.4.5.	204 No Content	27
4.4.6.	205 Reset Content	28
4.5.	Redirection 3xx	28
4.5.1.	300 Multiple Choices	29
4.5.2.	301 Moved Permanently	30
4.5.3.	302 Found	30
4.5.4.	303 See Other	31
4.5.5.	305 Use Proxy	31
4.5.6.	306 (Unused)	31
4.5.7.	307 Temporary Redirect	32
4.6.	Client Error 4xx	32
4.6.1.	400 Bad Request	32
4.6.2.	402 Payment Required	32
4.6.3.	403 Forbidden	32
4.6.4.	404 Not Found	33
4.6.5.	405 Method Not Allowed	33
4.6.6.	406 Not Acceptable	33
4.6.7.	408 Request Timeout	33
4.6.8.	409 Conflict	34
4.6.9.	410 Gone	34
4.6.10.	411 Length Required	34
4.6.11.	413 Request Representation Too Large	35
4.6.12.	414 URI Too Long	35
4.6.13.	415 Unsupported Media Type	35
4.6.14.	417 Expectation Failed	35
4.6.15.	426 Upgrade Required	35
4.7.	Server Error 5xx	36
4.7.1.	500 Internal Server Error	36
4.7.2.	501 Not Implemented	36

4.7.3.	502 Bad Gateway	36
4.7.4.	503 Service Unavailable	36
4.7.5.	504 Gateway Timeout	37
4.7.6.	505 HTTP Version Not Supported	37
5.	Protocol Parameters	37
5.1.	Date/Time Formats	37
5.2.	Product Tokens	40
5.3.	Character Encodings (charset)	41
5.4.	Content Codings	41
5.4.1.	Content Coding Registry	42
5.5.	Media Types	42
5.5.1.	Canonicalization and Text Defaults	43
5.5.2.	Multipart Types	44
5.6.	Language Tags	44
6.	Payload	45
6.1.	Payload Header Fields	45
6.2.	Payload Body	45
7.	Representation	45
7.1.	Identifying the Resource Associated with a Representation	46
7.2.	Representation Header Fields	47
7.3.	Representation Data	48
8.	Content Negotiation	49
8.1.	Server-driven Negotiation	50
8.2.	Agent-driven Negotiation	51
9.	Header Field Definitions	52
9.1.	Accept	52
9.2.	Accept-Charset	54
9.3.	Accept-Encoding	55
9.4.	Accept-Language	56
9.5.	Allow	57
9.6.	Content-Encoding	57
9.7.	Content-Language	58
9.8.	Content-Location	59
9.9.	Content-Type	61
9.10.	Date	61
9.11.	Expect	62
9.12.	From	63
9.13.	Location	63
9.14.	Max-Forwards	65
9.15.	Referer	65
9.16.	Retry-After	66
9.17.	Server	66
9.18.	User-Agent	67
10.	IANA Considerations	67
10.1.	Method Registry	67
10.2.	Status Code Registry	68
10.3.	Header Field Registration	69

10.4. Content Coding Registry	70
11. Security Considerations	71
11.1. Transfer of Sensitive Information	71
11.2. Encoding Sensitive Information in URIs	72
11.3. Location Header Fields: Spoofing and Information Leakage	72
11.4. Security Considerations for CONNECT	73
11.5. Privacy Issues Connected to Accept Header Fields	73
12. Acknowledgments	74
13. References	74
13.1. Normative References	74
13.2. Informative References	75
Appendix A. Differences between HTTP and MIME	77
A.1. MIME-Version	78
A.2. Conversion to Canonical Form	78
A.3. Conversion of Date Formats	79
A.4. Introduction of Content-Encoding	79
A.5. No Content-Transfer-Encoding	79
A.6. MHTML and Line Length Limitations	80
Appendix B. Additional Features	80
Appendix C. Changes from RFC 2616	80
Appendix D. Imported ABNF	82
Appendix E. Collected ABNF	83
Appendix F. Change Log (to be removed by RFC Editor before publication)	85
F.1. Since RFC 2616	85
F.2. Since draft-ietf-httpbis-p2-semantics-00	86
F.3. Since draft-ietf-httpbis-p3-payload-00	86
F.4. Since draft-ietf-httpbis-p2-semantics-01	87
F.5. Since draft-ietf-httpbis-p3-payload-01	88
F.6. Since draft-ietf-httpbis-p2-semantics-02	88
F.7. Since draft-ietf-httpbis-p3-payload-02	89
F.8. Since draft-ietf-httpbis-p2-semantics-03	89
F.9. Since draft-ietf-httpbis-p3-payload-03	89
F.10. Since draft-ietf-httpbis-p2-semantics-04	90
F.11. Since draft-ietf-httpbis-p3-payload-04	90
F.12. Since draft-ietf-httpbis-p2-semantics-05	91
F.13. Since draft-ietf-httpbis-p3-payload-05	91
F.14. Since draft-ietf-httpbis-p2-semantics-06	91
F.15. Since draft-ietf-httpbis-p3-payload-06	92
F.16. Since draft-ietf-httpbis-p2-semantics-07	92
F.17. Since draft-ietf-httpbis-p3-payload-07	92
F.18. Since draft-ietf-httpbis-p2-semantics-08	93
F.19. Since draft-ietf-httpbis-p3-payload-08	93
F.20. Since draft-ietf-httpbis-p2-semantics-09	93
F.21. Since draft-ietf-httpbis-p3-payload-09	94
F.22. Since draft-ietf-httpbis-p2-semantics-10	94
F.23. Since draft-ietf-httpbis-p3-payload-10	95

F.24. Since draft-ietf-httpbis-p2-semantics-11	95
F.25. Since draft-ietf-httpbis-p3-payload-11	96
F.26. Since draft-ietf-httpbis-p2-semantics-12	96
F.27. Since draft-ietf-httpbis-p3-payload-12	97
F.28. Since draft-ietf-httpbis-p2-semantics-13	97
F.29. Since draft-ietf-httpbis-p3-payload-13	98
F.30. Since draft-ietf-httpbis-p2-semantics-14	98
F.31. Since draft-ietf-httpbis-p3-payload-14	98
F.32. Since draft-ietf-httpbis-p2-semantics-15	98
F.33. Since draft-ietf-httpbis-p3-payload-15	99
F.34. Since draft-ietf-httpbis-p2-semantics-16	99
F.35. Since draft-ietf-httpbis-p3-payload-16	99
F.36. Since draft-ietf-httpbis-p2-semantics-17	99
F.37. Since draft-ietf-httpbis-p3-payload-17	100
F.38. Since draft-ietf-httpbis-p2-semantics-18	100
F.39. Since draft-ietf-httpbis-p3-payload-18	101
F.40. Since draft-ietf-httpbis-p2-semantics-19 and draft-ietf-httpbis-p3-payload-19	101
Index	101

1. Introduction

Each HTTP message is either a request or a response. A server listens on a connection for a request, parses each message received, interprets the message semantics in relation to the identified request target, and responds to that request with one or more response messages. This document defines HTTP/1.1 request and response semantics in terms of the architecture, syntax notation, and conformance criteria defined in [Part1].

HTTP provides a uniform interface for interacting with resources regardless of their type, nature, or implementation. HTTP semantics includes the intentions defined by each request method, extensions to those semantics that might be described in request header fields, the meaning of status codes to indicate a machine-readable response, and additional control data and resource metadata that might be given in response header fields.

In addition, this document defines the payload of messages (a.k.a., content), the associated metadata header fields that define how the payload is intended to be interpreted by a recipient, the request header fields that might influence content selection, and the various selection algorithms that are collectively referred to as "content negotiation".

Note: This document is currently disorganized in order to minimize changes between drafts and enable reviewers to see the smaller errata changes. A future draft will reorganize the sections to better reflect the content. In particular, the sections will be ordered according to the typical processing of an HTTP request message (after message parsing): resource mapping, methods, request modifying header fields, response status, status modifying header fields, and resource metadata. The current mess reflects how widely dispersed these topics and associated requirements had become in [RFC2616].

1.1. Conformance and Error Handling

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].

This specification targets conformance criteria according to the role of a participant in HTTP communication. Hence, HTTP requirements are placed on senders, recipients, clients, servers, user agents, intermediaries, origin servers, proxies, gateways, or caches, depending on what behavior is being constrained by the requirement. See Section 2 of [Part1] for definitions of these terms.

The verb "generate" is used instead of "send" where a requirement differentiates between creating a protocol element and merely forwarding a received element downstream.

An implementation is considered conformant if it complies with all of the requirements associated with the roles it partakes in HTTP. Note that SHOULD-level requirements are relevant here, unless one of the documented exceptions is applicable.

This document also uses ABNF to define valid protocol elements (Section 1.2). In addition to the prose requirements placed upon them, senders MUST NOT generate protocol elements that do not match the grammar defined by the ABNF rules for those protocol elements that are applicable to the sender's role. If a received protocol element is processed, the recipient MUST be able to parse any value that would match the ABNF rules for that protocol element, excluding only those rules not applicable to the recipient's role.

Unless noted otherwise, a recipient MAY attempt to recover a usable protocol element from an invalid construct. HTTP does not define specific error handling mechanisms except when they have a direct impact on security, since different applications of the protocol require different error handling strategies. For example, a Web browser might wish to transparently recover from a response where the Location header field doesn't parse according to the ABNF, whereas a systems control client might consider any form of error recovery to be dangerous.

1.2. Syntax Notation

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] with the list rule extension defined in Section 1.2 of [Part1]. Appendix D describes rules imported from other documents. Appendix E shows the collected ABNF with the list rule expanded.

2. Methods

The method token indicates the request method to be performed on the target resource (Section 5.5 of [Part1]). The method is case-sensitive.

method = token

The list of methods allowed by a resource can be specified in an Allow header field (Section 9.5). The status code of the response always notifies the client whether a method is currently allowed on a resource, since the set of allowed methods can change dynamically.

An origin server SHOULD respond with the status code 405 (Method Not Allowed) if the method is known by the origin server but not allowed for the resource, and 501 (Not Implemented) if the method is unrecognized or not implemented by the origin server. The methods GET and HEAD MUST be supported by all general-purpose servers. All other methods are OPTIONAL; however, if the above methods are implemented, they MUST be implemented with the same semantics as those specified in Section 2.3.

2.1. Safe and Idempotent Methods

2.1.1. Safe Methods

Implementers need to be aware that the software represents the user in their interactions over the Internet, and need to allow the user to be aware of any actions they take which might have an unexpected significance to themselves or others.

In particular, the convention has been established that the GET, HEAD, OPTIONS, and TRACE request methods SHOULD NOT have the significance of taking an action other than retrieval. These request methods ought to be considered "safe". This allows user agents to represent other methods, such as POST, PUT and DELETE, in a special way, so that the user is made aware of the fact that a possibly unsafe action is being requested.

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

2.1.2. Idempotent Methods

Request methods can also have the property of "idempotence" in that, aside from error or expiration issues, the intended effect of multiple identical requests is the same as for a single request. PUT, DELETE, and all safe request methods are idempotent. It is important to note that idempotence refers only to changes requested by the client: a server is free to change its state due to multiple requests for the purpose of tracking those requests, versioning of results, etc.

2.2. Method Registry

The HTTP Method Registry defines the name space for the method token in the Request line of an HTTP request.

Registrations MUST include the following fields:

- o Method Name (see Section 2)
- o Safe ("yes" or "no", see Section 2.1.1)
- o Idempotent ("yes" or "no", see Section 2.1.1)
- o Pointer to specification text

Values to be added to this name space require IETF Review (see [RFC5226], Section 4.1).

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

2.2.1. Considerations for New Methods

When it is necessary to express new semantics for a HTTP request that aren't specific to a single application or media type, and currently defined methods are inadequate, it might be appropriate to register a new method.

HTTP methods are generic; that is, they are potentially applicable to any resource, not just one particular media type, "type" of resource, or application. As such, it is preferred that new HTTP methods be registered in a document that isn't specific to a single application, so that this is clear.

Due to the parsing rules defined in Section 3.3 of [Part1], definitions of HTTP methods cannot prohibit the presence of a message body on either the request or the response message (with responses to HEAD requests being the single exception). Definitions of new methods cannot change this rule, but they can specify that only zero-length bodies (as opposed to absent bodies) are allowed.

New method definitions need to indicate whether they are safe (Section 2.1.1), what semantics (if any) the request body has, and whether they are idempotent (Section 2.1.2). They also need to state whether they can be cached ([Part6]); in particular under what conditions a cache can store the response, and under what conditions such a stored response can be used to satisfy a subsequent request.

2.3. Method Definitions

2.3.1. OPTIONS

The OPTIONS method requests information about the communication options available on the request/response chain identified by the effective request URI. This method allows a client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval.

Responses to the OPTIONS method are not cacheable.

If the OPTIONS request includes a message body (as indicated by the presence of Content-Length or Transfer-Encoding), then the media type MUST be indicated by a Content-Type field. Although this specification does not define any use for such a body, future extensions to HTTP might use the OPTIONS body to make more detailed queries on the server.

If the request-target (Section 5.3 of [Part1]) is an asterisk ("*"), the OPTIONS request is intended to apply to the server in general rather than to a specific resource. Since a server's communication options typically depend on the resource, the "*" request is only useful as a "ping" or "no-op" type of method; it does nothing beyond allowing the client to test the capabilities of the server. For example, this can be used to test a proxy for HTTP/1.1 conformance (or lack thereof).

If the request-target is not an asterisk, the OPTIONS request applies only to the options that are available when communicating with that resource.

A 200 (OK) response SHOULD include any header fields that indicate optional features implemented by the server and applicable to that resource (e.g., Allow), possibly including extensions not defined by this specification. The response body, if any, SHOULD also include information about the communication options. The format for such a body is not defined by this specification, but might be defined by future extensions to HTTP. Content negotiation MAY be used to select the appropriate response format. If no response body is included, the response MUST include a Content-Length field with a field-value of "0".

The Max-Forwards header field MAY be used to target a specific proxy in the request chain (see Section 9.14). If no Max-Forwards field is present in the request, then the forwarded request MUST NOT include a Max-Forwards field.

2.3.2. GET

The GET method requests transfer of a current representation of the target resource.

If the target resource is a data-producing process, it is the produced data which shall be returned as the representation in the response and not the source text of the process, unless that text happens to be the output of the process.

The semantics of the GET method change to a "conditional GET" if the request message includes an If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match, or If-Range header field ([Part4]). A conditional GET requests that the representation be transferred only under the circumstances described by the conditional header field(s). The conditional GET request is intended to reduce unnecessary network usage by allowing cached representations to be refreshed without requiring multiple requests or transferring data already held by the client.

The semantics of the GET method change to a "partial GET" if the request message includes a Range header field ([Part5]). A partial GET requests that only part of the representation be transferred, as described in Section 5.4 of [Part5]. The partial GET request is intended to reduce unnecessary network usage by allowing partially-retrieved representations to be completed without transferring data already held by the client.

Bodies on GET requests have no defined semantics. Note that sending a body on a GET request might cause some existing implementations to reject the request.

The response to a GET request is cacheable and MAY be used to satisfy subsequent GET and HEAD requests (see [Part6]).

See Section 11.2 for security considerations when used for forms.

2.3.3. HEAD

The HEAD method is identical to GET except that the server MUST NOT return a message body in the response. The metadata contained in the HTTP header fields in response to a HEAD request SHOULD be identical to the information sent in response to a GET request. This method can be used for obtaining metadata about the representation implied by the request without transferring the representation body. This method is often used for testing hypertext links for validity, accessibility, and recent modification.

The response to a HEAD request is cacheable and MAY be used to satisfy a subsequent HEAD request. It also has potential side effects on previously stored responses to GET; see Section 5 of [Part6].

Bodies on HEAD requests have no defined semantics. Note that sending a body on a HEAD request might cause some existing implementations to reject the request.

2.3.4. POST

The POST method requests that the origin server accept the representation enclosed in the request as data to be processed by the target resource. POST is designed to allow a uniform method to cover the following functions:

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

The actual function performed by the POST method is determined by the server and is usually dependent on the effective request URI.

The action performed by the POST method might not result in a resource that can be identified by a URI. In this case, either 200 (OK) or 204 (No Content) is the appropriate response status code, depending on whether or not the response includes a representation that describes the result.

If a resource has been created on the origin server, the response SHOULD be 201 (Created) and contain a representation which describes the status of the request and refers to the new resource, and a Location header field (see Section 9.13).

Responses to POST requests are only cacheable when they include explicit freshness information (see Section 4.1.1 of [Part6]). A cached POST response with a Content-Location header field (see Section 9.8) whose value is the effective Request URI MAY be used to satisfy subsequent GET and HEAD requests.

Note that POST caching is not widely implemented. However, the 303 (See Other) response can be used to direct the user agent to retrieve

a cacheable representation of the resource.

2.3.5. PUT

The PUT method requests that the state of the target resource be created or replaced with the state defined by the representation enclosed in the request message payload. A successful PUT of a given representation would suggest that a subsequent GET on that same target resource will result in an equivalent representation being returned in a 200 (OK) response. However, there is no guarantee that such a state change will be observable, since the target resource might be acted upon by other user agents in parallel, or might be subject to dynamic processing by the origin server, before any subsequent GET is received. A successful response only implies that the user agent's intent was achieved at the time of its processing by the origin server.

If the target resource does not have a current representation and the PUT successfully creates one, then the origin server **MUST** inform the user agent by sending a 201 (Created) response. If the target resource does have a current representation and that representation is successfully modified in accordance with the state of the enclosed representation, then either a 200 (OK) or 204 (No Content) response **SHOULD** be sent to indicate successful completion of the request.

Unrecognized header fields **SHOULD** be ignored (i.e., not saved as part of the resource state).

An origin server **SHOULD** verify that the PUT representation is consistent with any constraints which the server has for the target resource that cannot or will not be changed by the PUT. This is particularly important when the origin server uses internal configuration information related to the URI in order to set the values for representation metadata on GET responses. When a PUT representation is inconsistent with the target resource, the origin server **SHOULD** either make them consistent, by transforming the representation or changing the resource configuration, or respond with an appropriate error message containing sufficient information to explain why the representation is unsuitable. The 409 (Conflict) or 415 (Unsupported Media Type) status codes are suggested, with the latter being specific to constraints on Content-Type values.

For example, if the target resource is configured to always have a Content-Type of "text/html" and the representation being PUT has a Content-Type of "image/jpeg", then the origin server **SHOULD** do one of:

- a. reconfigure the target resource to reflect the new media type;
- b. transform the PUT representation to a format consistent with that of the resource before saving it as the new resource state; or,
- c. reject the request with a 415 (Unsupported Media Type) response indicating that the target resource is limited to "text/html", perhaps including a link to a different resource that would be a suitable target for the new representation.

HTTP does not define exactly how a PUT method affects the state of an origin server beyond what can be expressed by the intent of the user agent request and the semantics of the origin server response. It does not define what a resource might be, in any sense of that word, beyond the interface provided via HTTP. It does not define how resource state is "stored", nor how such storage might change as a result of a change in resource state, nor how the origin server translates resource state into representations. Generally speaking, all implementation details behind the resource interface are intentionally hidden by the server.

The fundamental difference between the POST and PUT methods is highlighted by the different intent for the target resource. The target resource in a POST request is intended to handle the enclosed representation as a data-accepting process, such as for a gateway to some other protocol or a document that accepts annotations. In contrast, the target resource in a PUT request is intended to take the enclosed representation as a new or replacement value. Hence, the intent of PUT is idempotent and visible to intermediaries, even though the exact effect is only known by the origin server.

Proper interpretation of a PUT request presumes that the user agent knows what target resource is desired. A service that is intended to select a proper URI on behalf of the client, after receiving a state-changing request, SHOULD be implemented using the POST method rather than PUT. If the origin server will not make the requested PUT state change to the target resource and instead wishes to have it applied to a different resource, such as when the resource has been moved to a different URI, then the origin server MUST send a 301 (Moved Permanently) response; the user agent MAY then make its own decision regarding whether or not to redirect the request.

A PUT request applied to the target resource MAY have side-effects on other resources. For example, an article might have a URI for identifying "the current version" (a resource) which is separate from the URIs identifying each particular version (different resources that at one point shared the same state as the current version resource). A successful PUT request on "the current version" URI

might therefore create a new version resource in addition to changing the state of the target resource, and might also cause links to be added between the related resources.

An origin server SHOULD reject any PUT request that contains a Content-Range header field (Section 5.2 of [Part5]), since it might be misinterpreted as partial content (or might be partial content that is being mistakenly PUT as a full representation). Partial content updates are possible by targeting a separately identified resource with state that overlaps a portion of the larger resource, or by using a different method that has been specifically defined for partial updates (for example, the PATCH method defined in [RFC5789]).

Responses to the PUT method are not cacheable. If a PUT request passes through a cache that has one or more stored responses for the effective request URI, those stored responses will be invalidated (see Section 6 of [Part6]).

2.3.6. DELETE

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

A successful response SHOULD be 200 (OK) if the response includes a representation describing the status, 202 (Accepted) if the action has not yet been enacted, or 204 (No Content) if the action has been enacted but the response does not include a representation.

Bodies on DELETE requests have no defined semantics. Note that sending a body on a DELETE request might cause some existing implementations to reject the request.

Responses to the DELETE method are not cacheable. If a DELETE request passes through a cache that has one or more stored responses for the effective request URI, those stored responses will be invalidated (see Section 6 of [Part6]).

2.3.7. TRACE

The TRACE method requests a remote, application-layer loop-back of the request message. The final recipient of the request SHOULD reflect the message received back to the client as the message body

of a 200 (OK) response. The final recipient is either the origin server or the first proxy to receive a Max-Forwards value of zero (0) in the request (see Section 9.14). A TRACE request MUST NOT include a message body.

TRACE allows the client to see what is being received at the other end of the request chain and use that data for testing or diagnostic information. The value of the Via header field (Section 6.2 of [Part1]) is of particular interest, since it acts as a trace of the request chain. Use of the Max-Forwards header field allows the client to limit the length of the request chain, which is useful for testing a chain of proxies forwarding messages in an infinite loop.

If the request is valid, the response SHOULD have a Content-Type of "message/http" (see Section 7.3.1 of [Part1]) and contain a message body that encloses a copy of the entire request message. Responses to the TRACE method are not cacheable.

2.3.8. CONNECT

The CONNECT method requests that the proxy establish a tunnel to the request-target and, if successful, thereafter restrict its behavior to blind forwarding of packets until the connection is closed.

When using CONNECT, the request-target MUST use the authority form (Section 5.3 of [Part1]); i.e., the request-target consists of only the host name and port number of the tunnel destination, separated by a colon. For example,

```
CONNECT server.example.com:80 HTTP/1.1
Host: server.example.com:80
```

Any 2xx (Successful) response to a CONNECT request indicates that the proxy has established a connection to the requested host and port, and has switched to tunneling the current connection to that server connection. The tunneled data from the server begins immediately after the blank line that concludes the successful response's header block.

A server SHOULD NOT send any Transfer-Encoding or Content-Length header fields in a successful response. A client MUST ignore any Content-Length or Transfer-Encoding header fields received in a successful response.

Any response other than a successful response indicates that the tunnel has not yet been formed and that the connection remains governed by HTTP.

Proxy authentication might be used to establish the authority to create a tunnel:

```
CONNECT server.example.com:80 HTTP/1.1
Host: server.example.com:80
Proxy-Authorization: basic aGVsbG86d29ybGQ=
```

A message body on a CONNECT request has no defined semantics. Sending a body on a CONNECT request might cause existing implementations to reject the request.

Similar to a pipelined HTTP/1.1 request, data to be tunneled from client to server MAY be sent immediately after the request (before a response is received). The usual caveats also apply: data can be discarded if the eventual response is negative, and the connection can be reset with no response if more than one TCP segment is outstanding.

It might be the case that the proxy itself can only reach the requested origin server through another proxy. In this case, the first proxy SHOULD make a CONNECT request of that next proxy, requesting a tunnel to the authority. A proxy MUST NOT respond with any 2xx status code unless it has either a direct or tunnel connection established to the authority.

If at any point either one of the peers gets disconnected, any outstanding data that came from that peer will be passed to the other one, and after that also the other connection will be terminated by the proxy. If there is outstanding data to that peer undelivered, that data will be discarded.

An origin server which receives a CONNECT request for itself MAY respond with a 2xx status code to indicate that a connection is established. However, most origin servers do not implement CONNECT.

3. Header Fields

Header fields are key value pairs that can be used to communicate data about the message, its payload, the target resource, or about the connection itself (i.e., control data). See Section 3.2 of [Part1] for a general definition of their syntax.

3.1. Considerations for Creating Header Fields

New header fields are registered using the procedures described in [RFC3864].

The requirements for header field names are defined in Section 4.1 of [RFC3864]. Authors of specifications defining new fields are advised to keep the name as short as practical, and not to prefix them with "X-" if they are to be registered (either immediately or in the future).

New header field values typically have their syntax defined using ABNF ([RFC5234]), using the extension defined in Appendix B of [Part1] as necessary, and are usually constrained to the range of ASCII characters. Header fields needing a greater range of characters can use an encoding such as the one defined in [RFC5987].

Because commas (",") are used as a generic delimiter between field-values, they need to be treated with care if they are allowed in the field-value's payload. Typically, components that might contain a comma are protected with double-quotes using the quoted-string ABNF production (Section 3.2.4 of [Part1]).

For example, a textual date and a URI (either of which might contain a comma) could be safely carried in field-values like these:

```
Example-URI-Field: "http://example.com/a.html,foo",  
                  "http://without-a-comma.example.com/"  
Example-Date-Field: "Sat, 04 May 1996", "Wed, 14 Sep 2005"
```

Note that double quote delimiters almost always are used with the quoted-string production; using a different syntax inside double quotes will likely cause unnecessary confusion.

Many header fields use a format including (case-insensitively) named parameters (for instance, Content-Type, defined in Section 9.9). Allowing both unquoted (token) and quoted (quoted-string) syntax for the parameter value enables recipients to use existing parser components. When allowing both forms, the meaning of a parameter value ought to be independent of the syntax used for it (for an example, see the notes on parameter handling for media types in Section 5.5).

Authors of specifications defining new header fields are advised to consider documenting:

- o Whether the field is a single value, or whether it can be a list (delimited by commas; see Section 3.2 of [Part1]).

If it does not use the list syntax, document how to treat messages where the header field occurs multiple times (a sensible default would be to ignore the header field, but this might not always be the right choice).

Note that intermediaries and software libraries might combine multiple header field instances into a single one, despite the header field not allowing this. A robust format enables recipients to discover these situations (good example: "Content-Type", as the comma can only appear inside quoted strings; bad example: "Location", as a comma can occur inside a URI).

- o Under what conditions the header field can be used; e.g., only in responses or requests, in all messages, only on responses to a particular request method.
- o Whether it is appropriate to list the field-name in the Connection header field (i.e., if the header field is to be hop-by-hop, see Section 6.1 of [Part1]).
- o Under what conditions intermediaries are allowed to modify the header field's value, insert or delete it.
- o How the header field might interact with caching (see [Part6]).
- o Whether the header field is useful or allowable in trailers (see Section 4.1 of [Part1]).
- o Whether the header field ought to be preserved across redirects.

3.2. Request Header Fields

The request header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers, with semantics equivalent to the parameters on a programming language method invocation.

Header Field Name	Defined in...
Accept	Section 9.1
Accept-Charset	Section 9.2
Accept-Encoding	Section 9.3
Accept-Language	Section 9.4
Authorization	Section 4.1 of [Part7]
Expect	Section 9.11
From	Section 9.12
Host	Section 5.4 of [Part1]
If-Match	Section 3.1 of [Part4]
If-Modified-Since	Section 3.3 of [Part4]
If-None-Match	Section 3.2 of [Part4]
If-Range	Section 5.3 of [Part5]
If-Unmodified-Since	Section 3.4 of [Part4]
Max-Forwards	Section 9.14
Proxy-Authorization	Section 4.3 of [Part7]
Range	Section 5.4 of [Part5]
Referer	Section 9.15
TE	Section 4.3 of [Part1]
User-Agent	Section 9.18

3.3. Response Header Fields

The response header fields allow the server to pass additional information about the response which cannot be placed in the status-line. These header fields give information about the server and about further access to the target resource (Section 5.5 of [Part1]).

Header Field Name	Defined in...
Accept-Ranges	Section 5.1 of [Part5]
Age	Section 7.1 of [Part6]
Allow	Section 9.5
Date	Section 9.10
ETag	Section 2.3 of [Part4]
Location	Section 9.13
Proxy-Authenticate	Section 4.2 of [Part7]
Retry-After	Section 9.16
Server	Section 9.17
Vary	Section 7.5 of [Part6]
WWW-Authenticate	Section 4.4 of [Part7]

4. Status Codes

The status-code element is a 3-digit integer result code of the attempt to understand and satisfy the request.

HTTP status codes are extensible. HTTP applications are not required to understand the meaning of all registered status codes, though such understanding is obviously desirable. However, applications **MUST** understand the class of any status code, as indicated by the first digit, and treat any unrecognized response as being equivalent to the x00 status code of that class, with the exception that an unrecognized response **MUST NOT** be cached. For example, if an unrecognized status code of 431 is received by the client, it can safely assume that there was something wrong with its request and treat the response as if it had received a 400 status code. In such cases, user agents **SHOULD** present to the user the representation enclosed with the response, since that representation is likely to include human-readable information which will explain the unusual status.

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:

- o 1xx (Informational): Request received, continuing process
- o 2xx (Successful): The action was successfully received, understood, and accepted
- o 3xx (Redirection): Further action needs to be taken in order to complete the request
- o 4xx (Client Error): The request contains bad syntax or cannot be fulfilled
- o 5xx (Server Error): The server failed to fulfill an apparently valid request

For most status codes the response can carry a payload, in which case a Content-Type header field indicates the payload's media type (Section 9.9).

4.1. Overview of Status Codes

The status codes listed below are defined in this specification, Section 4 of [Part4], Section 3 of [Part5], and Section 3 of [Part7]. The reason phrases listed here are only recommendations -- they can be replaced by local equivalents without affecting the protocol.

status-code	reason-phrase	Defined in...
100	Continue	Section 4.3.1
101	Switching Protocols	Section 4.3.2
200	OK	Section 4.4.1
201	Created	Section 4.4.2
202	Accepted	Section 4.4.3
203	Non-Authoritative Information	Section 4.4.4
204	No Content	Section 4.4.5
205	Reset Content	Section 4.4.6
206	Partial Content	Section 3.1 of [Part5]
300	Multiple Choices	Section 4.5.1
301	Moved Permanently	Section 4.5.2
302	Found	Section 4.5.3
303	See Other	Section 4.5.4
304	Not Modified	Section 4.1 of [Part4]
305	Use Proxy	Section 4.5.5
307	Temporary Redirect	Section 4.5.7
400	Bad Request	Section 4.6.1
401	Unauthorized	Section 3.1 of [Part7]
402	Payment Required	Section 4.6.2
403	Forbidden	Section 4.6.3
404	Not Found	Section 4.6.4
405	Method Not Allowed	Section 4.6.5
406	Not Acceptable	Section 4.6.6
407	Proxy Authentication Required	Section 3.2 of [Part7]
408	Request Time-out	Section 4.6.7
409	Conflict	Section 4.6.8
410	Gone	Section 4.6.9
411	Length Required	Section 4.6.10
412	Precondition Failed	Section 4.2 of [Part4]
413	Request Representation Too Large	Section 4.6.11
414	URI Too Long	Section 4.6.12
415	Unsupported Media Type	Section 4.6.13
416	Requested range not satisfiable	Section 3.2 of [Part5]
417	Expectation Failed	Section 4.6.14
426	Upgrade Required	Section 4.6.15
500	Internal Server Error	Section 4.7.1
501	Not Implemented	Section 4.7.2

502	Bad Gateway	Section 4.7.3	
503	Service Unavailable	Section 4.7.4	
504	Gateway Time-out	Section 4.7.5	
505	HTTP Version not supported	Section 4.7.6	
+-----+-----+-----+-----+			

Note that this list is not exhaustive -- it does not include extension status codes defined in other specifications.

4.2. Status Code Registry

The HTTP Status Code Registry defines the name space for the status-code token in the status-line of an HTTP response.

Values to be added to this name space require IETF Review (see [RFC5226], Section 4.1).

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

4.2.1. Considerations for New Status Codes

When it is necessary to express new semantics for a HTTP response that aren't specific to a single application or media type, and currently defined status codes are inadequate, a new status code can be registered.

HTTP status codes are generic; that is, they are potentially applicable to any resource, not just one particular media type, "type" of resource, or application. As such, it is preferred that new HTTP status codes be registered in a document that isn't specific to a single application, so that this is clear.

Definitions of new HTTP status codes typically explain the request conditions that produce a response containing the status code (e.g., combinations of request header fields and/or method(s)), along with any interactions with response header fields (e.g., those that are required, those that modify the semantics of the response).

New HTTP status codes are required to fall under one of the categories defined in Section 4. To allow existing parsers to properly handle them, new status codes cannot disallow a response body, although they can mandate a zero-length response body. They can require the presence of one or more particular HTTP response header field(s).

Likewise, their definitions can specify that caches are allowed to use heuristics to determine their freshness (see [Part6]; by default,

it is not allowed), and can define how to determine the resource which they carry a representation for (see Section 7.1; by default, it is anonymous).

4.3. Informational lxx

This class of status code indicates a provisional response, consisting only of the status-line and optional header fields, and is terminated by an empty line. There are no required header fields for this class of status code. Since HTTP/1.0 did not define any lxx status codes, servers **MUST NOT** send a lxx response to an HTTP/1.0 client except under experimental conditions.

A client **MUST** be prepared to accept one or more lxx status responses prior to a regular response, even if the client does not expect a 100 (Continue) status message. Unexpected lxx status responses **MAY** be ignored by a user agent.

Proxies **MUST** forward lxx responses, unless the connection between the proxy and its client has been closed, or unless the proxy itself requested the generation of the lxx response. (For example, if a proxy adds an "Expect: 100-continue" field when it forwards a request, then it need not forward the corresponding 100 (Continue) response(s).)

4.3.1. 100 Continue

The client **SHOULD** continue with its request. This interim response is used to inform the client that the initial part of the request has been received and has not yet been rejected by the server. The client **SHOULD** continue by sending the remainder of the request or, if the request has already been completed, ignore this response. The server **MUST** send a final response after the request has been completed. See Section 6.4.3 of [Part1] for detailed discussion of the use and handling of this status code.

4.3.2. 101 Switching Protocols

The server understands and is willing to comply with the client's request, via the Upgrade message header field (Section 6.5 of [Part1]), for a change in the application protocol being used on this connection. The server will switch protocols to those defined by the response's Upgrade header field immediately after the empty line which terminates the 101 response.

The protocol **SHOULD** be switched only when it is advantageous to do so. For example, switching to a newer version of HTTP is advantageous over older versions, and switching to a real-time,

synchronous protocol might be advantageous when delivering resources that use such features.

4.4. Successful 2xx

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

4.4.1. 200 OK

The request has succeeded. The payload returned with the response is dependent on the method used in the request, for example:

GET a representation of the target resource is sent in the response;

HEAD the same representation as GET, except without the message body;

POST a representation describing or containing the result of the action;

TRACE a representation containing the request message as received by the end server.

Caches MAY use a heuristic (see Section 4.1.2 of [Part6]) to determine freshness for 200 responses.

4.4.2. 201 Created

The request has been fulfilled and has resulted in one or more new resources being created.

Newly created resources are typically linked to from the response payload, with the most relevant URI also being carried in the Location header field. If the newly created resource's URI is the same as the Effective Request URI, this information can be omitted (e.g., in the case of a response to a PUT request).

The origin server MUST create the resource(s) before returning the 201 status code. If the action cannot be carried out immediately, the server SHOULD respond with 202 (Accepted) response instead.

A 201 response MAY contain an ETag response header field indicating the current value of the entity-tag for the representation of the resource identified by the Location header field or, in case the Location header field was omitted, by the Effective Request URI (see Section 2.3 of [Part4]).

4.4.3. 202 Accepted

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

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

4.4.4. 203 Non-Authoritative Information

The representation in the response has been transformed or otherwise modified by a transforming proxy (Section 2.4 of [Part1]). Note that the behavior of transforming intermediaries is controlled by the no-transform Cache-Control directive (Section 7.2 of [Part6]).

This status code is only appropriate when the response status code would have been 200 (OK) otherwise. When the status code before transformation would have been different, the 214 Transformation Applied warn-code (Section 7.6 of [Part6]) is appropriate.

Caches MAY use a heuristic (see Section 4.1.2 of [Part6]) to determine freshness for 203 responses.

4.4.5. 204 No Content

The 204 (No Content) status code indicates that the server has successfully fulfilled the request and that there is no additional content to return in the response payload body. Metadata in the response header fields refer to the target resource and its current representation after the requested action.

For example, if a 204 status code is received in response to a PUT request and the response contains an ETag header field, then the PUT was successful and the ETag field-value contains the entity-tag for the new representation of that target resource.

The 204 response allows a server to indicate that the action has been successfully applied to the target resource while implying that the user agent SHOULD NOT traverse away from its current "document view"

(if any). The server assumes that the user agent will provide some indication of the success to its user, in accord with its own interface, and apply any new or updated metadata in the response to the active representation.

For example, a 204 status code is commonly used with document editing interfaces corresponding to a "save" action, such that the document being saved remains available to the user for editing. It is also frequently used with interfaces that expect automated data transfers to be prevalent, such as within distributed version control systems.

The 204 response **MUST NOT** include a message body, and thus is always terminated by the first empty line after the header fields.

4.4.6. 205 Reset Content

The server has fulfilled the request and the user agent **SHOULD** reset the document view which caused the request to be sent. This response is primarily intended to allow input for actions to take place via user input, followed by a clearing of the form in which the input is given so that the user can easily initiate another input action.

The message body included with the response **MUST** be empty. Note that receivers still need to parse the response according to the algorithm defined in Section 3.3 of [Part1].

4.5. Redirection 3xx

This class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request. If the required action involves a subsequent HTTP request, it **MAY** be carried out by the user agent without interaction with the user if and only if the method used in the second request is known to be "safe", as defined in Section 2.1.1.

There are several types of redirects:

1. Redirects of the request to another URI, either temporarily or permanently. The new URI is specified in the Location header field. In this specification, the status codes 301 (Moved Permanently), 302 (Found), and 307 (Temporary Redirect) fall under this category.
2. Redirection to a new location that represents an indirect response to the request, such as the result of a POST operation to be retrieved with a subsequent GET request. This is status code 303 (See Other).

3. Redirection offering a choice of matching resources for use by agent-driven content negotiation (Section 8.2). This is status code 300 (Multiple Choices).
4. Other kinds of redirection, such as to a cached result (status code 304 (Not Modified), see Section 4.1 of [Part4]).

Note: In HTTP/1.0, only the status codes 301 (Moved Permanently) and 302 (Found) were defined for the first type of redirect, and the second type did not exist at all ([RFC1945], Section 9.3). However it turned out that web forms using POST expected redirects to change the operation for the subsequent request to retrieval (GET). To address this use case, HTTP/1.1 introduced the second type of redirect with the status code 303 (See Other) ([RFC2068], Section 10.3.4). As user agents did not change their behavior to maintain backwards compatibility, the first revision of HTTP/1.1 added yet another status code, 307 (Temporary Redirect), for which the backwards compatibility problems did not apply ([RFC2616], Section 10.3.8). Over 10 years later, most user agents still do method rewriting for status codes 301 and 302, therefore this specification makes that behavior conformant in case the original request was POST.

A Location header field on a 3xx response indicates that a client MAY automatically redirect to the URI provided; see Section 9.13.

Note that for methods not known to be "safe", as defined in Section 2.1.1, automatic redirection needs to be done with care, since the redirect might change the conditions under which the request was issued.

Clients SHOULD detect and intervene in cyclical redirections (i.e., "infinite" redirection loops).

Note: An earlier version of this specification recommended a maximum of five redirections ([RFC2068], Section 10.3). Content developers need to be aware that some clients might implement such a fixed limitation.

4.5.1. 300 Multiple Choices

The target resource has more than one representation, each with its own specific location, and agent-driven negotiation information (Section 8) is being provided so that the user (or user agent) can select a preferred representation by redirecting its request to that location.

Unless it was a HEAD request, the response SHOULD include a

representation containing a list of representation metadata and location(s) from which the user or user agent can choose the one most appropriate. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice MAY be performed automatically. However, this specification does not define any standard for such automatic selection.

If the server has a preferred choice of representation, it SHOULD include the specific URI for that representation in the Location field; user agents MAY use the Location field value for automatic redirection.

Caches MAY use a heuristic (see Section 4.1.2 of [Part6]) to determine freshness for 300 responses.

4.5.2. 301 Moved Permanently

The target resource has been assigned a new permanent URI and any future references to this resource SHOULD use one of the returned URIs. Clients with link editing capabilities ought to automatically re-link references to the effective request URI to one or more of the new references returned by the server, where possible.

Caches MAY use a heuristic (see Section 4.1.2 of [Part6]) to determine freshness for 301 responses.

The new permanent URI SHOULD be given by the Location field in the response. A response payload can contain a short hypertext note with a hyperlink to the new URI(s).

Note: For historic reasons, user agents MAY change the request method from POST to GET for the subsequent request. If this behavior is undesired, status code 307 (Temporary Redirect) can be used instead.

4.5.3. 302 Found

The target resource resides temporarily under a different URI. Since the redirection might be altered on occasion, the client SHOULD continue to use the effective request URI for future requests.

The temporary URI SHOULD be given by the Location field in the response. A response payload can contain a short hypertext note with a hyperlink to the new URI(s).

Note: For historic reasons, user agents MAY change the request method from POST to GET for the subsequent request. If this behavior is undesired, status code 307 (Temporary Redirect) can be

used instead.

4.5.4. 303 See Other

The 303 status code indicates that the server is redirecting the user agent to a different resource, as indicated by a URI in the Location header field, that is intended to provide an indirect response to the original request. In order to satisfy the original request, a user agent SHOULD perform a retrieval request using the Location URI (a GET or HEAD request if using HTTP), which can itself be redirected further, and present the eventual result as an answer to the original request. Note that the new URI in the Location header field is not considered equivalent to the effective request URI.

This status code is generally applicable to any HTTP method. It is primarily used to allow the output of a POST action to redirect the user agent to a selected resource, since doing so provides the information corresponding to the POST response in a form that can be separately identified, bookmarked, and cached independent of the original request.

A 303 response to a GET request indicates that the requested resource does not have a representation of its own that can be transferred by the server over HTTP. The Location URI indicates a resource that is descriptive of the target resource, such that the follow-on representation might be useful to recipients without implying that it adequately represents the target resource. Note that answers to the questions of what can be represented, what representations are adequate, and what might be a useful description are outside the scope of HTTP and thus entirely determined by the URI owner(s).

Except for responses to a HEAD request, the representation of a 303 response SHOULD contain a short hypertext note with a hyperlink to the Location URI.

4.5.5. 305 Use Proxy

The 305 status code was defined in a previous version of this specification (see Appendix C), and is now deprecated.

4.5.6. 306 (Unused)

The 306 status code was used in a previous version of the specification, is no longer used, and the code is reserved.

4.5.7. 307 Temporary Redirect

The target resource resides temporarily under a different URI. Since the redirection can change over time, the client **SHOULD** continue to use the effective request URI for future requests.

The temporary URI **SHOULD** be given by the Location field in the response. A response payload can contain a short hypertext note with a hyperlink to the new URI(s).

Note: This status code is similar to 302 (Found), except that it does not allow rewriting the request method from POST to GET. This specification defines no equivalent counterpart for 301 (Moved Permanently) ([draft-reschke-http-status-308], however, defines the status code 308 (Permanent Redirect) for this purpose).

4.6. Client Error 4xx

The 4xx class of status code is intended for cases in which the client seems to have erred. Except when responding to a HEAD request, the server **SHOULD** include a representation containing an explanation of the error situation, and whether it is a temporary or permanent condition. These status codes are applicable to any request method. User agents **SHOULD** display any included representation to the user.

4.6.1. 400 Bad Request

The server cannot or will not process the request, due to a client error (e.g., malformed syntax).

4.6.2. 402 Payment Required

This code is reserved for future use.

4.6.3. 403 Forbidden

The server understood the request, but refuses to authorize it. Providing different user authentication credentials might be successful, but any credentials that were provided in the request are insufficient. The request **SHOULD NOT** be repeated with the same credentials.

If the request method was not HEAD and the server wishes to make public why the request has not been fulfilled, it **SHOULD** describe the reason for the refusal in the representation. If the server does not wish to make this information available to the client, the status

code 404 (Not Found) MAY be used instead.

4.6.4. 404 Not Found

The server has not found anything matching the effective request URI. No indication is given of whether the condition is temporary or permanent. The 410 (Gone) status code SHOULD be used if the server knows, through some internally configurable mechanism, that an old resource is permanently unavailable and has no forwarding address. This status code is commonly used when the server does not wish to reveal exactly why the request has been refused, or when no other response is applicable.

4.6.5. 405 Method Not Allowed

The method specified in the request-line is not allowed for the target resource. The response MUST include an Allow header field containing a list of valid methods for the requested resource.

4.6.6. 406 Not Acceptable

The resource identified by the request is only capable of generating response representations which have content characteristics not acceptable according to the Accept and Accept-* header fields sent in the request.

Unless it was a HEAD request, the response SHOULD include a representation containing a list of available representation characteristics and location(s) from which the user or user agent can choose the one most appropriate. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice MAY be performed automatically. However, this specification does not define any standard for such automatic selection.

Note: HTTP/1.1 servers are allowed to return responses which are not acceptable according to the accept header fields sent in the request. In some cases, this might even be preferable to sending a 406 response. User agents are encouraged to inspect the header fields of an incoming response to determine if it is acceptable.

If the response could be unacceptable, a user agent SHOULD temporarily stop receipt of more data and query the user for a decision on further actions.

4.6.7. 408 Request Timeout

The client did not produce a request within the time that the server was prepared to wait. The client MAY repeat the request without

modifications at any later time.

4.6.8. 409 Conflict

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

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

4.6.9. 410 Gone

The target resource is no longer available at the server and no forwarding address is known. This condition is expected to be considered permanent. Clients with link editing capabilities SHOULD delete references to the effective request URI after user approval. If the server does not know, or has no facility to determine, whether or not the condition is permanent, the status code 404 (Not Found) SHOULD be used instead.

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

Caches MAY use a heuristic (see Section 4.1.2 of [Part6]) to determine freshness for 410 responses.

4.6.10. 411 Length Required

The server refuses to accept the request without a defined Content-Length. The client MAY repeat the request if it adds a valid

Content-Length header field containing the length of the message body in the request message.

4.6.11. 413 Request Representation Too Large

The server is refusing to process a request because the request representation is larger than the server is willing or able to process. The server MAY close the connection to prevent the client from continuing the request.

If the condition is temporary, the server SHOULD include a Retry-After header field to indicate that it is temporary and after what time the client MAY try again.

4.6.12. 414 URI Too Long

The server is refusing to service the request because the effective request URI is longer than the server is willing to interpret. This rare condition is only likely to occur when a client has improperly converted a POST request to a GET request with long query information, when the client has descended into a URI "black hole" of redirection (e.g., a redirected URI prefix that points to a suffix of itself), or when the server is under attack by a client attempting to exploit security holes present in some servers using fixed-length buffers for reading or manipulating the request-target.

4.6.13. 415 Unsupported Media Type

The server is refusing to service the request because the request payload is in a format not supported by this request method on the target resource.

4.6.14. 417 Expectation Failed

The expectation given in an Expect header field (see Section 9.11) could not be met by this server, or, if the server is a proxy, the server has unambiguous evidence that the request could not be met by the next-hop server.

4.6.15. 426 Upgrade Required

The request can not be completed without a prior protocol upgrade. This response MUST include an Upgrade header field (Section 6.5 of [Part1]) specifying the required protocols.

Example:

```
HTTP/1.1 426 Upgrade Required
Upgrade: HTTP/3.0
Connection: Upgrade
Content-Length: 53
Content-Type: text/plain
```

This service requires use of the HTTP/3.0 protocol.

The server SHOULD include a message body in the 426 response which indicates in human readable form the reason for the error and describes any alternative courses which might be available to the user.

4.7. Server Error 5xx

Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has erred or is incapable of performing the request. Except when responding to a HEAD request, the server SHOULD include a representation containing an explanation of the error situation, and whether it is a temporary or permanent condition. User agents SHOULD display any included representation to the user. These response codes are applicable to any request method.

4.7.1. 500 Internal Server Error

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

4.7.2. 501 Not Implemented

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

4.7.3. 502 Bad Gateway

The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request.

4.7.4. 503 Service Unavailable

The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.

The implication is that this is a temporary condition which will be alleviated after some delay. If known, the length of the delay MAY be indicated in a Retry-After header field (Section 9.16). If no Retry-After is given, the client SHOULD handle the response as it would for a 500 (Internal Server Error) response.

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

4.7.5. 504 Gateway Timeout

The server, while acting as a gateway or proxy, did not receive a timely response from the upstream server specified by the URI (e.g., HTTP, FTP, LDAP) or some other auxiliary server (e.g., DNS) it needed to access in attempting to complete the request.

Note to implementers: some deployed proxies are known to return 400 (Bad Request) or 500 (Internal Server Error) when DNS lookups time out.

4.7.6. 505 HTTP Version Not Supported

The server does not support, or refuses to support, the protocol version that was used in the request message. The server is indicating that it is unable or unwilling to complete the request using the same major version as the client, as described in Section 2.7 of [Part1], other than with this error message. The response SHOULD contain a representation describing why that version is not supported and what other protocols are supported by that server.

5. Protocol Parameters

5.1. Date/Time Formats

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.

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, day, month, year, and time-of-day 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-1 "," SP date2 SP time-of-day SP GMT
date2        = day "-" month "-" 2DIGIT
               ; day-month-year (e.g., 02-Jun-82)

day-name-1    = %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.

5.2. 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).

5.3. Character Encodings (charset)

HTTP uses charset names to indicate the character encoding of a textual representation.

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

charset = token

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

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

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

5.4. Content Codings

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

content-coding = token

All content-coding values are case-insensitive. HTTP/1.1 uses content-coding values in the Accept-Encoding (Section 9.3) and Content-Encoding (Section 9.6) header fields. Although the value describes the content-coding, what is more important is that it indicates what decoding mechanism will be required to remove the encoding.

compress

See Section 4.2.1 of [Part1].

deflate

See Section 4.2.2 of [Part1].

gzip

See Section 4.2.3 of [Part1].

5.4.1. Content Coding Registry

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

Registrations MUST include the following fields:

- o Name
- o Description
- o Pointer to specification text

Names of content codings MUST NOT overlap with names of transfer codings (Section 4 of [Part1]), unless the encoding transformation is identical (as is the case for the compression codings defined in Section 4.2 of [Part1]).

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

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

5.5. Media Types

HTTP uses Internet Media Types [RFC2046] in the Content-Type (Section 9.9) and Accept (Section 9.1) header fields in order to provide open and extensible data typing and type negotiation.

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

The type/subtype MAY be followed by parameters in the form of

attribute/value pairs.

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

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

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

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

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

5.5.1. Canonicalization and Text Defaults

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

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

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

5.5.2. Multipart Types

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

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

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

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

5.6. Language Tags

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

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

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

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

Example tags include:

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

See [RFC5646] for further information.

6. Payload

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

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

6.1. Payload Header Fields

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

Header Field Name	Defined in...
Content-Length	Section 3.3.2 of [Part1]
Content-Range	Section 5.2 of [Part5]

6.2. Payload Body

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

7. Representation

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

at some point in the future (e.g., in a PUT request).

Most, but not all, representations transferred via HTTP are intended to be a representation of the target resource (the resource identified by the effective request URI). The precise semantics of a representation are determined by the type of message (request or response), the request method, the response status code, and the representation metadata. For example, the above semantic is true for the representation in any 200 (OK) response to GET and for the representation in any PUT request. A 200 response to PUT, in contrast, contains either a representation that describes the successful action or a representation of the target resource, with the latter indicated by a Content-Location header field with the same value as the effective request URI. Likewise, response messages with an error status code usually contain a representation that describes the error and what next steps are suggested for resolving it.

Request and Response messages MAY transfer a representation if not otherwise restricted by the request method or response status code. A representation consists of metadata (representation header fields) and data (representation body). When a complete or partial representation is enclosed in an HTTP message, it is referred to as the payload of the message.

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

7.1. Identifying the Resource Associated with a Representation

It is sometimes necessary to determine an identifier for the resource associated with a representation.

An HTTP request representation, when present, is always associated with an anonymous (i.e., unidentified) resource.

In the common case, an HTTP response is a representation of the target resource (see Section 5.5 of [Part1]). However, this is not always the case. To determine the URI of the resource a response is associated with, the following rules are used (with the first applicable one being selected):

1. If the response status code is 200 (OK) or 203 (Non-Authoritative Information) and the request method was GET, the response payload is a representation of the target resource.

2. If the response status code is 204 (No Content), 206 (Partial Content), or 304 (Not Modified) and the request method was GET or HEAD, the response payload is a partial representation of the target resource.
3. If the response has a Content-Location header field, and that URI is the same as the effective request URI, the response payload is a representation of the target resource.
4. If the response has a Content-Location header field, and that URI is not the same as the effective request URI, then the response asserts that its payload is a representation of the resource identified by the Content-Location URI. However, such an assertion cannot be trusted unless it can be verified by other means (not defined by HTTP).
5. Otherwise, the response is a representation of an anonymous (i.e., unidentified) resource.

[[TODO-req-uri: The comparison function is going to have to be defined somewhere, because we already need to compare URIs for things like cache invalidation.]]

7.2. Representation Header Fields

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

The following header fields are defined as representation metadata:

Header Field Name	Defined in...
Content-Encoding	Section 9.6
Content-Language	Section 9.7
Content-Location	Section 9.8
Content-Type	Section 9.9
Expires	Section 7.3 of [Part6]

We use the term "selected representation" to refer to the the current representation of a target resource that would have been selected in a successful response if the same request had used the method GET and excluded any conditional request header fields.

Additional header fields define metadata about the selected representation, which might differ from the representation included in the message for responses to some state-changing methods. The following header fields are defined as selected representation metadata:

Header Field Name	Defined in...
ETag	Section 2.3 of [Part4]
Last-Modified	Section 2.2 of [Part4]

7.3. Representation Data

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

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

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

Content-Type specifies the media type of the underlying data, which defines both the data format and how that data SHOULD be processed by the recipient (within the scope of the request method semantics). Any HTTP/1.1 message containing a payload body SHOULD include a Content-Type header field defining the media type of the associated representation unless that metadata is unknown to the sender. If the Content-Type header field is not present, it indicates that the sender does not know the media type of the representation; recipients MAY either assume that the media type is "application/octet-stream" ([RFC2046], Section 4.5.1) or examine the content to determine its type.

In practice, resource owners do not always properly configure their origin server to provide the correct Content-Type for a given representation, with the result that some clients will examine a response body's content and override the specified type. Clients that do so risk drawing incorrect conclusions, which might expose additional security risks (e.g., "privilege escalation"). Furthermore, it is impossible to determine the sender's intent by examining the data format: many data formats match multiple media types that differ only in processing semantics. Implementers are

encouraged to provide a means of disabling such "content sniffing" when it is used.

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

8. Content Negotiation

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

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

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

These patterns are all widely used, and have trade-offs in applicability and practicality. In particular, when the number of preferences or capabilities to be expressed by a client are large (such as when many different formats are supported by a user-agent), server-driven negotiation becomes unwieldy, and might not be appropriate. Conversely, when the number of representations to choose from is very large, agent-driven negotiation might not be appropriate.

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

8.1. Server-driven Negotiation

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

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

Server-driven negotiation has disadvantages:

1. It is impossible for the server to accurately determine what might be "best" for any given user, since that would require complete knowledge of both the capabilities of the user agent and the intended use for the response (e.g., does the user want to view it on screen or print it on paper?).
2. Having the user agent describe its capabilities in every request can be both very inefficient (given that only a small percentage of responses have multiple representations) and a potential violation of the user's privacy.
3. It complicates the implementation of an origin server and the algorithms for generating responses to a request.
4. It might limit a public cache's ability to use the same response for multiple user's requests.

Server-driven negotiation allows the user agent to specify its preferences, but it cannot expect responses to always honor them. For example, the origin server might not implement server-driven negotiation, or it might decide that sending a response that doesn't conform to them is better than sending a 406 (Not Acceptable) response.

Many of the mechanisms for expressing preferences use quality values to declare relative preference. See Section 4.3.1 of [Part1] for

more information.

HTTP/1.1 includes the following header fields for enabling server-driven negotiation through description of user agent capabilities and user preferences: Accept (Section 9.1), Accept-Charset (Section 9.2), Accept-Encoding (Section 9.3), Accept-Language (Section 9.4), and User-Agent (Section 9.18). However, an origin server is not limited to these dimensions and MAY vary the response based on any aspect of the request, including aspects of the connection (e.g., IP address) or information within extension header fields not defined by this specification.

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

The Vary header field (Section 7.5 of [Part6]) can be used to express the parameters the server uses to select a representation that is subject to server-driven negotiation.

8.2. Agent-driven Negotiation

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

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

Agent-driven negotiation suffers from the disadvantage of needing a second request to obtain the best alternate representation. This second request is only efficient when caching is used. In addition, this specification does not define any mechanism for supporting automatic selection, though it also does not prevent any such mechanism from being developed as an extension and used within HTTP/1.1.

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

server-driven negotiation.

9. Header Field Definitions

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

9.1. Accept

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

```
Accept = #( media-range [ accept-params ] )

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

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

Each media-range MAY be followed by one or more accept-params, beginning with the "q" parameter for indicating a relative quality factor. The first "q" parameter (if any) separates the media-range parameter(s) from the accept-params. Quality factors allow the user or user agent to indicate the relative degree of preference for that media-range, using the qvalue scale from 0 to 1 (Section 4.3.1 of [Part1]). The default value is q=1.

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

The example

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

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

A request without any Accept header field implies that the user agent will accept any media type in response. If an Accept header field is present in a request and none of the available representations for the response have a media type that is listed as acceptable, the origin server MAY either honor the Accept header field by sending a 406 (Not Acceptable) response or disregard the Accept header field by treating the response as if it is not subject to content negotiation.

A more elaborate example is

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

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

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

Accept: text/*, text/plain, text/plain;format=flowed, */*

have the following precedence:

1. text/plain;format=flowed
2. text/plain
3. text/*
4. */*

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

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

would cause the following values to be associated:

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

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

9.2. Accept-Charset

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

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

Character encoding values (a.k.a., charsets) are described in Section 5.3. Each charset MAY be given an associated quality value which represents the user's preference for that charset. The default value is q=1. An example is

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

The special value "*", if present in the Accept-Charset field, matches every character encoding which is not mentioned elsewhere in the Accept-Charset field. If no "*" is present in an Accept-Charset field, then all character encodings not explicitly mentioned get a quality value of 0.

A request without any Accept-Charset header field implies that the user agent will accept any character encoding in response. If an Accept-Charset header field is present in a request and none of the available representations for the response have a character encoding that is listed as acceptable, the origin server MAY either honor the Accept-Charset header field by sending a 406 (Not Acceptable) response or disregard the Accept-Charset header field by treating the

response as if it is not subject to content negotiation.

9.3. Accept-Encoding

The "Accept-Encoding" header field can be used by user agents to indicate what response content-codings (Section 5.4) are acceptable in the response. An "identity" token is used as a synonym for "no encoding" in order to communicate when no encoding is preferred.

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

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

For example,

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

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

1. The special "*" symbol in an Accept-Encoding field matches any available content-coding not explicitly listed in the header field.
2. If the representation has no content-coding, then it is acceptable by default unless specifically excluded by the Accept-Encoding field stating either "identity;q=0" or "*;q=0" without a more specific entry for "identity".
3. If the representation's content-coding is one of the content-codings listed in the Accept-Encoding field, then it is acceptable unless it is accompanied by a qvalue of 0. (As defined in Section 4.3.1 of [Part1], a qvalue of 0 means "not acceptable".)
4. If multiple content-codings are acceptable, then the acceptable content-coding with the highest non-zero qvalue is preferred.

An Accept-Encoding header field with a combined field-value that is empty implies that the user agent does not want any content-coding in response. If an Accept-Encoding header field is present in a request

and none of the available representations for the response have a content-coding that is listed as acceptable, the origin server SHOULD send a response without any content-coding.

A request without an Accept-Encoding header field implies that the user agent will accept any content-coding in response, but a representation without content-coding is preferred for compatibility with the widest variety of user agents.

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

9.4. Accept-Language

The "Accept-Language" header field can be used by user agents to indicate the set of natural languages that are preferred in the response. Language tags are defined in Section 5.6.

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

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

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

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

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

Note: The "Basic Filtering" scheme ([RFC4647], Section 3.3.1) is identical to the matching scheme that was previously defined in Section 14.4 of [RFC2616].

It might be contrary to the privacy expectations of the user to send an Accept-Language header field with the complete linguistic preferences of the user in every request. For a discussion of this issue, see Section 11.5.

As intelligibility is highly dependent on the individual user, it is

recommended that client applications make the choice of linguistic preference available to the user. If the choice is not made available, then the Accept-Language header field **MUST NOT** be given in the request.

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

9.5. Allow

The "Allow" header field lists the set of methods advertised as supported by the target resource. The purpose of this field is strictly to inform the recipient of valid request methods associated with the resource.

Allow = #method

Example of use:

Allow: GET, HEAD, PUT

The actual set of allowed methods is defined by the origin server at the time of each request.

A proxy **MUST NOT** modify the Allow header field -- it does not need to understand all the methods specified in order to handle them according to the generic message handling rules.

9.6. Content-Encoding

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

Content-Encoding = 1#content-coding

Content codings are defined in Section 5.4. An example of its use is

Content-Encoding: gzip

The content-coding is a characteristic of the representation. Typically, the representation body is stored with this encoding and is only decoded before rendering or analogous usage. However, a transforming proxy MAY modify the content-coding if the new coding is known to be acceptable to the recipient, unless the "no-transform" cache-control directive is present in the message.

If the media type includes an inherent encoding, such as a data format that is always compressed, then that encoding would not be restated as a Content-Encoding even if it happens to be the same algorithm as one of the content-codings. Such a content-coding would only be listed if, for some bizarre reason, it is applied a second time to form the representation. Likewise, an origin server might choose to publish the same payload data as multiple representations that differ only in whether the coding is defined as part of Content-Type or Content-Encoding, since some user agents will behave differently in their handling of each response (e.g., open a "Save as ..." dialog instead of automatic decompression and rendering of content).

A representation that has a content-coding applied to it MUST include a Content-Encoding header field that lists the content-coding(s) applied.

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

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

9.7. Content-Language

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

Content-Language = 1#language-tag

Language tags are defined in Section 5.6. The primary purpose of Content-Language is to allow a user to identify and differentiate representations according to the user's own preferred language. Thus, if the body content is intended only for a Danish-literate

audience, the appropriate field is

Content-Language: da

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

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

Content-Language: mi, en

However, just because multiple languages are present within a representation does not mean that it is intended for multiple linguistic audiences. An example would be a beginner's language primer, such as "A First Lesson in Latin", which is clearly intended to be used by an English-literate audience. In this case, the Content-Language would properly only include "en".

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

9.8. Content-Location

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

Content-Location = absolute-URI / partial-URI

The Content-Location value is not a replacement for the effective Request URI (Section 5.5 of [Part1]). It is representation metadata. It has the same syntax and semantics as the header field of the same name defined for MIME body parts in Section 4 of [RFC2557]. However, its appearance in an HTTP message has some special implications for HTTP recipients.

If Content-Location is included in a response message and its value is the same as the effective request URI, then the response payload SHOULD be considered a current representation of that resource. For a GET or HEAD request, this is the same as the default semantics when no Content-Location is provided by the server. For a state-changing

request like PUT or POST, it implies that the server's response contains the new representation of that resource, thereby distinguishing it from representations that might only report about the action (e.g., "It worked!"). This allows authoring applications to update their local copies without the need for a subsequent GET request.

If Content-Location is included in a response message and its value differs from the effective request URI, then the origin server is informing recipients that this representation has its own, presumably more specific, identifier. For a GET or HEAD request, this is an indication that the effective request URI identifies a resource that is subject to content negotiation and the selected representation for this response can also be found at the identified URI. For other methods, such a Content-Location indicates that this representation contains a report on the action's status and the same report is available (for future access with GET) at the given URI. For example, a purchase transaction made via a POST request might include a receipt document as the payload of the 200 (OK) response; the Content-Location value provides an identifier for retrieving a copy of that same receipt in the future.

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

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

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

If the Content-Location value is a partial URI, the partial URI is interpreted relative to the effective request URI.

9.9. Content-Type

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

Content-Type = media-type

Media types are defined in Section 5.5. An example of the field is

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

Further discussion of Content-Type is provided in Section 7.3.

9.10. 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 defined in Section 5.1; it MUST be sent in rfc1123-date format.

Date = 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.

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 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.

9.11. Expect

The "Expect" header field is used to indicate that particular server behaviors are required by the client.

```
Expect          = 1#expectation

expectation     = expect-name [ BWS "=" BWS expect-value ]
                  *( OWS ";" [ OWS expect-param ] )
expect-param    = expect-name [ BWS "=" BWS expect-value ]

expect-name     = token
expect-value    = token / quoted-string
```

If all received Expect header field(s) are syntactically valid but contain an expectation that the recipient does not understand or cannot comply with, the recipient MUST respond with a 417 (Expectation Failed) status code. A recipient of a syntactically invalid Expectation header field MUST respond with a 4xx status code other than 417.

The only expectation defined by this specification is:

100-continue

The "100-continue" expectation is defined Section 6.4.3 of [Part1]. It does not support any expect-params.

Comparison is case-insensitive for names (expect-name), and case-sensitive for values (expect-value).

The Expect mechanism is hop-by-hop: the above requirements apply to any server, including proxies. However, the Expect header field itself is end-to-end; it MUST be forwarded if the request is forwarded.

Many older HTTP/1.0 and HTTP/1.1 applications do not understand the Expect header field.

9.12. From

The "From" header field, if given, SHOULD contain an Internet e-mail address for the human user who controls the requesting user agent. The address SHOULD be machine-usable, as defined by "mailbox" in Section 3.4 of [RFC5322]:

From = mailbox

mailbox = <mailbox, defined in [RFC5322], Section 3.4>

An example is:

From: webmaster@example.org

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

The Internet e-mail address in this field MAY be separate from the Internet host which issued the request. For example, when a request is passed through a proxy the original issuer's address SHOULD be used.

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

9.13. Location

The "Location" header field MAY be sent in responses to refer to a specific resource in accordance with the semantics of the status code.

Location = URI-reference

For 201 (Created) responses, the Location is the URI of the new resource which was created by the request. For 3xx (Redirection) responses, the location SHOULD indicate the server's preferred URI for automatic redirection to the resource.

The field value consists of a single URI-reference. When it has the form of a relative reference ([RFC3986], Section 4.2), the final value is computed by resolving it against the effective request URI ([RFC3986], Section 5). If the original URI, as navigated to by the user agent, did contain a fragment identifier, and the final value does not, then the original URI's fragment identifier is added to the final value.

For example, the original URI "http://www.example.org/~tim", combined with a field value given as:

Location: /pub/WWW/People.html#tim

would result in a final value of
"http://www.example.org/pub/WWW/People.html#tim"

An original URI "http://www.example.org/index.html#larry", combined with a field value given as:

Location: http://www.example.net/index.html

would result in a final value of
"http://www.example.net/index.html#larry", preserving the original fragment identifier.

Note: Some recipients attempt to recover from Location fields that are not valid URI references. This specification does not mandate or define such processing, but does allow it.

There are circumstances in which a fragment identifier in a Location URI would not be appropriate. For instance, when it appears in a 201 (Created) response, where the Location header field specifies the URI for the entire created resource.

Note: The Content-Location header field (Section 9.8) differs from Location in that the Content-Location identifies the most specific resource corresponding to the enclosed representation. It is therefore possible for a response to contain header fields for both Location and Content-Location.

9.14. Max-Forwards

The "Max-Forwards" header field provides a mechanism with the TRACE (Section 2.3.7) and OPTIONS (Section 2.3.1) methods to limit the number of times that the request is forwarded by proxies. This can be useful when the client is attempting to trace a request which appears to be failing or looping mid-chain.

Max-Forwards = 1*DIGIT

The Max-Forwards value is a decimal integer indicating the remaining number of times this request message can be forwarded.

Each recipient of a TRACE or OPTIONS request containing a Max-Forwards header field MUST check and update its value prior to forwarding the request. If the received value is zero (0), the recipient MUST NOT forward the request; instead, it MUST respond as the final recipient. If the received Max-Forwards value is greater than zero, then the forwarded message MUST contain an updated Max-Forwards field with a value decremented by one (1).

The Max-Forwards header field MAY be ignored for all other request methods.

9.15. Referer

The "Referer" [sic] header field allows the client to specify the URI of the resource from which the target URI was obtained (the "referrer", although the header field is misspelled.).

The Referer header field allows servers to generate lists of back-links to resources for interest, logging, optimized caching, etc. It also allows obsolete or mistyped links to be traced for maintenance. Some servers use Referer as a means of controlling where they allow links from (so-called "deep linking"), but legitimate requests do not always contain a Referer header field.

If the target URI was obtained from a source that does not have its own URI (e.g., input from the user keyboard), the Referer field MUST either be sent with the value "about:blank", or not be sent at all. Note that this requirement does not apply to sources with non-HTTP URIs (e.g., FTP).

Referer = absolute-URI / partial-URI

Example:

Referer: http://www.example.org/hypertext/Overview.html

If the field value is a relative URI, it SHOULD be interpreted relative to the effective request URI. The URI MUST NOT include a fragment. See Section 11.2 for security considerations.

9.16. Retry-After

The header "Retry-After" field can be used with a 503 (Service Unavailable) response to indicate how long the service is expected to be unavailable to the requesting client. This field MAY also be used with any 3xx (Redirection) response to indicate the minimum time the user-agent is asked to wait before issuing the redirected request.

The value of this field can be either an HTTP-date or an integer number of seconds (in decimal) after the time of the response.

Retry-After = HTTP-date / delta-seconds

Time spans are non-negative decimal integers, representing time in seconds.

delta-seconds = 1*DIGIT

Two examples of its use are

Retry-After: Fri, 31 Dec 1999 23:59:59 GMT
Retry-After: 120

In the latter example, the delay is 2 minutes.

9.17. Server

The "Server" header field contains information about the software used by the origin server to handle the request.

The field can contain multiple product tokens (Section 5.2) and comments (Section 3.2 of [Part1]) identifying the server and any significant subproducts. The product tokens are listed in order of their significance for identifying the application.

Server = product *(RWS (product / comment))

Example:

Server: CERN/3.0 libwww/2.17

If the response is being forwarded through a proxy, the proxy application MUST NOT modify the Server header field. Instead, it MUST include a Via field (as described in Section 6.2 of [Part1]).

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

9.18. User-Agent

The "User-Agent" header field contains information about the user agent originating the request. User agents SHOULD include this field with requests.

Typically, it is used for statistical purposes, the tracing of protocol violations, and tailoring responses to avoid particular user agent limitations.

The field can contain multiple product tokens (Section 5.2) and comments (Section 3.2 of [Part1]) identifying the agent and its significant subproducts. By convention, the product tokens are listed in order of their significance for identifying the application.

Because this field is usually sent on every request a user agent makes, implementations are encouraged not to include needlessly fine-grained detail, and to limit (or even prohibit) the addition of subproducts by third parties. Overly long and detailed User-Agent field values make requests larger and can also be used to identify ("fingerprint") the user against their wishes.

Likewise, implementations are encouraged not to use the product tokens of other implementations in order to declare compatibility with them, as this circumvents the purpose of the field. Finally, they are encouraged not to use comments to identify products; doing so makes the field value more difficult to parse.

User-Agent = product *(RWS (product / comment))

Example:

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

10. IANA Considerations

10.1. Method Registry

The registration procedure for HTTP request methods is defined by Section 2.2 of this document.

The HTTP Method Registry shall be created at <http://www.iana.org/assignments/http-methods> and be populated with the registrations below:

Method	Safe	Idempotent	Reference
CONNECT	no	no	Section 2.3.8
DELETE	no	yes	Section 2.3.6
GET	yes	yes	Section 2.3.2
HEAD	yes	yes	Section 2.3.3
OPTIONS	yes	yes	Section 2.3.1
POST	no	no	Section 2.3.4
PUT	no	yes	Section 2.3.5
TRACE	yes	yes	Section 2.3.7

10.2. Status Code Registry

The registration procedure for HTTP Status Codes -- previously defined in Section 7.1 of [RFC2817] -- is now defined by Section 4.2 of this document.

The HTTP Status Code Registry located at <http://www.iana.org/assignments/http-status-codes> shall be updated with the registrations below:

Value	Description	Reference
100	Continue	Section 4.3.1
101	Switching Protocols	Section 4.3.2
200	OK	Section 4.4.1
201	Created	Section 4.4.2
202	Accepted	Section 4.4.3
203	Non-Authoritative Information	Section 4.4.4
204	No Content	Section 4.4.5
205	Reset Content	Section 4.4.6
300	Multiple Choices	Section 4.5.1
301	Moved Permanently	Section 4.5.2
302	Found	Section 4.5.3
303	See Other	Section 4.5.4
305	Use Proxy	Section 4.5.5
306	(Unused)	Section 4.5.6
307	Temporary Redirect	Section 4.5.7
400	Bad Request	Section 4.6.1
402	Payment Required	Section 4.6.2
403	Forbidden	Section 4.6.3
404	Not Found	Section 4.6.4
405	Method Not Allowed	Section 4.6.5
406	Not Acceptable	Section 4.6.6
408	Request Timeout	Section 4.6.7
409	Conflict	Section 4.6.8
410	Gone	Section 4.6.9
411	Length Required	Section 4.6.10
413	Request Representation Too Large	Section 4.6.11
414	URI Too Long	Section 4.6.12
415	Unsupported Media Type	Section 4.6.13
417	Expectation Failed	Section 4.6.14
426	Upgrade Required	Section 4.6.15
500	Internal Server Error	Section 4.7.1
501	Not Implemented	Section 4.7.2
502	Bad Gateway	Section 4.7.3
503	Service Unavailable	Section 4.7.4
504	Gateway Timeout	Section 4.7.5
505	HTTP Version Not Supported	Section 4.7.6

10.3. Header Field Registration

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

Header Field Name	Protocol	Status	Reference
Accept	http	standard	Section 9.1
Accept-Charset	http	standard	Section 9.2
Accept-Encoding	http	standard	Section 9.3
Accept-Language	http	standard	Section 9.4
Allow	http	standard	Section 9.5
Content-Encoding	http	standard	Section 9.6
Content-Language	http	standard	Section 9.7
Content-Location	http	standard	Section 9.8
Content-Type	http	standard	Section 9.9
Date	http	standard	Section 9.10
Expect	http	standard	Section 9.11
From	http	standard	Section 9.12
Location	http	standard	Section 9.13
MIME-Version	http	standard	Appendix A.1
Max-Forwards	http	standard	Section 9.14
Referer	http	standard	Section 9.15
Retry-After	http	standard	Section 9.16
Server	http	standard	Section 9.17
User-Agent	http	standard	Section 9.18

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

10.4. Content Coding Registry

The registration procedure for HTTP Content Codings is now defined by Section 5.4.1 of this document.

The HTTP Content Codings Registry located at <http://www.iana.org/assignments/http-parameters> shall be updated with the registration below:

Name	Description	Reference
compress	UNIX "compress" program method	Section 4.2.1 of [Part1]
deflate	"deflate" compression mechanism ([RFC1951]) used inside the "zlib" data format ([RFC1950])	Section 4.2.2 of [Part1]
gzip	Same as GNU zip [RFC1952]	Section 4.2.3 of [Part1]

identity	reserved (synonym for "no encoding" in	Section 9.3
	Accept-Encoding header field)	
+-----+-----+-----+-----+		

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. Transfer of Sensitive Information

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

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

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

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

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

We suggest, though do not require, that a convenient toggle interface

be provided for the user to enable or disable the sending of From and Referer information.

The User-Agent (Section 9.18) or Server (Section 9.17) header fields can sometimes be used to determine that a specific client or server has a particular security hole which might be exploited. Unfortunately, this same information is often used for other valuable purposes for which HTTP currently has no better mechanism.

Furthermore, the User-Agent header field might contain enough entropy to be used, possibly in conjunction with other material, to uniquely identify the user.

Some request methods, like TRACE (Section 2.3.7), expose information that was sent in request header fields within the body of their response. Clients SHOULD be careful with sensitive information, like Cookies, Authorization credentials, and other header fields that might be used to collect data from the client.

11.2. Encoding Sensitive Information in URIs

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

Clients SHOULD NOT include a Referer header field in a (non-secure) HTTP request if the referring page was transferred with a secure protocol.

Authors of services SHOULD NOT use GET-based forms for the submission of sensitive data because that data will be placed in the request-target. Many existing servers, proxies, and user agents log or display the request-target in places where it might be visible to third parties. Such services can use POST-based form submission instead.

11.3. Location Header Fields: Spoofing and Information Leakage

If a single server supports multiple organizations that do not trust one another, then it MUST check the values of Location and Content-Location header fields in responses that are generated under control of said organizations to make sure that they do not attempt to invalidate resources over which they have no authority.

Furthermore, appending the fragment identifier from one URI to another one obtained from a Location header field might leak confidential information to the target server -- although the fragment identifier is not transmitted in the final request, it might be visible to the user agent through other means, such as scripting.

11.4. Security Considerations for CONNECT

Since tunneled data is opaque to the proxy, there are additional risks to tunneling to other well-known or reserved ports. A HTTP client CONNECTing to port 25 could relay spam via SMTP, for example. As such, proxies SHOULD restrict CONNECT access to a small number of known ports.

11.5. Privacy Issues Connected to Accept Header Fields

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

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

Elaborate user-customized accept header fields sent in every request, in particular if these include quality values, can be used by servers as relatively reliable and long-lived user identifiers. Such user identifiers would allow content providers to do click-trail tracking, and would allow collaborating content providers to match cross-server click-trails or form submissions of individual users. Note that for many users not behind a proxy, the network address of the host running the user agent will also serve as a long-lived user identifier. In environments where proxies are used to enhance privacy, user agents ought to be conservative in offering accept header field configuration options to end users. As an extreme privacy measure, proxies could filter the accept header fields in relayed requests. General purpose user agents which provide a high degree of header field configurability SHOULD warn users about the loss of privacy which can be involved.

12. Acknowledgments

See Section 9 of [Part1].

13. References

13.1. Normative References

- [Part1] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: Message Routing and Syntax", draft-ietf-httpbis-p1-messaging-20 (work in progress), July 2012.
- [Part4] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 4: Conditional Requests", draft-ietf-httpbis-p4-conditional-20 (work in progress), July 2012.
- [Part5] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 5: Range Requests", draft-ietf-httpbis-p5-range-20 (work in progress), July 2012.
- [Part6] Fielding, R., Ed., Lafon, Y., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1, part 6: Caching", draft-ietf-httpbis-p6-cache-20 (work in progress), July 2012.
- [Part7] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 7: Authentication", draft-ietf-httpbis-p7-auth-20 (work in progress), July 2012.
- [RFC1950] Deutsch, L. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, May 1996.
- [RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.
- [RFC1952] Deutsch, P., Gailly, J-L., Adler, M., Deutsch, L., and G. Randers-

- Pehrson, "GZIP file format specification version 4.3", RFC 1952, May 1996.
- [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.
- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4647] Phillips, A., Ed. and M. Davis, Ed., "Matching of Language Tags", BCP 47, RFC 4647, September 2006.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, September 2009.

13.2. Informative References

- [RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.
- [RFC1945] Berners-Lee, T., Fielding, R., and H. Nielsen, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945,

May 1996.

- [RFC2049] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples", RFC 2049, November 1996.
- [RFC2068] Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997.
- [RFC2076] Palme, J., "Common Internet Message Headers", RFC 2076, February 1997.
- [RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.
- [RFC2295] Holtman, K. and A. Mutz, "Transparent Content Negotiation in HTTP", RFC 2295, March 1998.
- [RFC2388] Masinter, L., "Returning Values from Forms: multipart/form-data", RFC 2388, August 1998.
- [RFC2557] Palme, F., Hopmann, A., Shelness, N., and E. Stefferud, "MIME Encapsulation of Aggregate Documents, such as HTML (MHTML)", RFC 2557, March 1999.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC2817] Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", RFC 2817, May 2000.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

- [RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.
- [RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC5322] Resnick, P., "Internet Message Format", RFC 5322, October 2008.
- [RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, March 2010.
- [RFC5987] Reschke, J., "Character Set and Language Encoding for Hypertext Transfer Protocol (HTTP) Header Field Parameters", RFC 5987, August 2010.
- [RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, March 2011.
- [RFC6266] Reschke, J., "Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol (HTTP)", RFC 6266, June 2011.
- [draft-reschke-http-status-308] Reschke, J., "The Hypertext Transfer Protocol (HTTP) Status Code 308 (Permanent Redirect)", draft-reschke-http-status-308-07 (work in progress), March 2012.

Appendix A. Differences between HTTP and MIME

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

[RFC2045]) to allow a message body to be transmitted in an open variety of representations and with extensible mechanisms. However, RFC 2045 discusses mail, and HTTP has a few features that are different from those described in MIME. These differences were carefully chosen to optimize performance over binary connections, to allow greater freedom in the use of new media types, to make date comparisons easier, and to acknowledge the practice of some early HTTP servers and clients.

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

A.1. MIME-Version

HTTP is not a MIME-compliant protocol. However, HTTP/1.1 messages MAY include a single MIME-Version header field to indicate what version of the MIME protocol was used to construct the message. Use of the MIME-Version header field indicates that the message is in full conformance with the MIME protocol (as defined in [RFC2045]). Proxies/gateways are responsible for ensuring full conformance (where possible) when exporting HTTP messages to strict MIME environments.

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

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

A.2. Conversion to Canonical Form

MIME requires that an Internet mail body-part be converted to canonical form prior to being transferred, as described in Section 4 of [RFC2049]. Section 5.5.1 of this document describes the forms allowed for subtypes of the "text" media type when transmitted over HTTP. [RFC2046] requires that content with a type of "text" represent line breaks as CRLF and forbids the use of CR or LF outside of line break sequences. HTTP allows CRLF, bare CR, and bare LF to indicate a line break within text content when a message is transmitted over HTTP.

Where it is possible, a proxy or gateway from HTTP to a strict MIME environment SHOULD translate all line breaks within the text media types described in Section 5.5.1 of this document to the RFC 2049 canonical form of CRLF. Note, however, that this might be

complicated by the presence of a Content-Encoding and by the fact that HTTP allows the use of some character encodings which do not use octets 13 and 10 to represent CR and LF, respectively, as is the case for some multi-byte character encodings.

Conversion will break any cryptographic checksums applied to the original content unless the original content is already in canonical form. Therefore, the canonical form is recommended for any content that uses such checksums in HTTP.

A.3. Conversion of Date Formats

HTTP/1.1 uses a restricted set of date formats (Section 5.1) to simplify the process of date comparison. Proxies and gateways from other protocols SHOULD ensure that any Date header field present in a message conforms to one of the HTTP/1.1 formats and rewrite the date if necessary.

A.4. Introduction of Content-Encoding

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

A.5. No Content-Transfer-Encoding

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

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

A.6. MHTML and Line Length Limitations

HTTP implementations which share code with MHTML [RFC2557] implementations need to be aware of MIME line length limitations. Since HTTP does not have this limitation, HTTP does not fold long lines. MHTML messages being transported by HTTP follow all conventions of MHTML, including line length limitations and folding, canonicalization, etc., since HTTP transports all message-bodies as payload (see Section 5.5.2) and does not interpret the content or any MIME header lines that might be contained therein.

Appendix B. Additional Features

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

A number of other header fields, such as Content-Disposition and Title, from SMTP and MIME are also often implemented (see [RFC6266] and [RFC2076]).

Appendix C. Changes from RFC 2616

Introduce Method Registry. (Section 2.2)

Clarify definition of POST. (Section 2.3.4)

Remove requirement to handle all Content-* header fields; ban use of Content-Range with PUT. (Section 2.3.5)

Take over definition of CONNECT method from [RFC2817]. (Section 2.3.8)

Take over the Status Code Registry, previously defined in Section 7.1 of [RFC2817]. (Section 4.2)

Broadened the definition of 203 (Non-Authoritative Information) to include cases of payload transformations as well. (Section 4.4.4)

Status codes 301, 302, and 307: removed the normative requirements on both response payloads and user interaction. (Section 4.5)

Failed to consider that there are many other request methods that are

safe to automatically redirect, and further that the user agent is able to make that determination based on the request method semantics. Furthermore, allow user agents to rewrite the method from POST to GET for status codes 301 and 302. (Sections 4.5.2, 4.5.3 and 4.5.7)

Deprecate 305 (Use Proxy) status code, because user agents did not implement it. It used to indicate that the target resource needs to be accessed through the proxy given by the Location field. The Location field gave the URI of the proxy. The recipient was expected to repeat this single request via the proxy. (Section 4.5.5)

Define status 426 (Upgrade Required) (this was incorporated from [RFC2817]). (Section 4.6.15)

Change ABNF productions for header fields to only define the field value. (Section 9)

Reclassify "Allow" as response header field, removing the option to specify it in a PUT request. Relax the server requirement on the contents of the Allow header field and remove requirement on clients to always trust the header field value. (Section 9.5)

The ABNF for the Expect header field has been both fixed (allowing parameters for value-less expectations as well) and simplified (allowing trailing semicolons after "100-continue" when they were invalid before). (Section 9.11)

Correct syntax of Location header field to allow URI references (including relative references and fragments), as referred symbol "absoluteURI" wasn't what was expected, and add some clarifications as to when use of fragments would not be appropriate. (Section 9.13)

Restrict Max-Forwards header field to OPTIONS and TRACE (previously, extension methods could have used it as well). (Section 9.14)

Allow Referer field value of "about:blank" as alternative to not specifying it. (Section 9.15)

In the description of the Server header field, the Via field was described as a SHOULD. The requirement was and is stated correctly in the description of the Via header field in Section 6.2 of [Part1]. (Section 9.17)

Clarify contexts that charset is used in. (Section 5.3)

Registration of Content Codings now requires IETF Review (Section 5.4.1)

Remove the default character encoding of "ISO-8859-1" for text media types; the default now is whatever the media type definition says. (Section 5.5.1)

Change ABNF productions for header fields to only define the field value. (Section 9)

Remove definition of Content-MD5 header field because it was inconsistently implemented with respect to partial responses, and also because of known deficiencies in the hash algorithm itself (see [RFC6151] for details). (Section 9)

Remove ISO-8859-1 special-casing in Accept-Charset. (Section 9.2)

Remove base URI setting semantics for Content-Location due to poor implementation support, which was caused by too many broken servers emitting bogus Content-Location header fields, and also the potentially undesirable effect of potentially breaking relative links in content-negotiated resources. (Section 9.8)

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

Remove discussion of Content-Disposition header field, it is now defined by [RFC6266]. (Appendix B)

Appendix D. Imported ABNF

The following core rules are included by reference, as defined in Appendix B.1 of [RFC5234]: 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), HTAB (horizontal tab), LF (line feed), OCTET (any 8-bit sequence of data), SP (space), and VCHAR (any visible US-ASCII character).

The rules below are defined in [Part1]:

BWS	= <BWS, defined in [Part1], Section 3.2.1>
OWS	= <OWS, defined in [Part1], Section 3.2.1>
RWS	= <RWS, defined in [Part1], Section 3.2.1>
quoted-string	= <quoted-string, defined in [Part1], Section 3.2.4>
token	= <token, defined in [Part1], Section 3.2.4>
word	= <word, defined in [Part1], Section 3.2.4>
absolute-URI	= <absolute-URI, defined in [Part1], Section 2.8>
comment	= <comment, defined in [Part1], Section 3.2.4>
partial-URI	= <partial-URI, defined in [Part1], Section 2.8>
qvalue	= <qvalue, defined in [Part1], Section 4.3.1>

URI-reference = <URI-reference, defined in [Part1], Section 2.8>

Appendix E. Collected ABNF

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

BWS = <BWS, defined in [Part1], Section 3.2.1>

Content-Encoding = *( "," OWS ) content-coding *( OWS "," [ OWS
    content-coding ] )
Content-Language = *( "," OWS ) language-tag *( OWS "," [ OWS
    language-tag ] )
Content-Location = absolute-URI / partial-URI
Content-Type = media-type

Date = HTTP-date

Expect = *( "," OWS ) expectation *( OWS "," [ OWS expectation ] )

From = mailbox

GMT = %x47.4D.54 ; GMT

HTTP-date = rfc1123-date / obs-date

Location = URI-reference

MIME-Version = 1*DIGIT "." 1*DIGIT
Max-Forwards = 1*DIGIT

OWS = <OWS, defined in [Part1], Section 3.2.1>

RWS = <RWS, defined in [Part1], Section 3.2.1>
Referer = absolute-URI / partial-URI
Retry-After = HTTP-date / delta-seconds

Server = product *( RWS ( product / comment ) )
```

URI-reference = <URI-reference, defined in [Part1], Section 2.8>
User-Agent = product *(RWS (product / comment))

absolute-URI = <absolute-URI, defined in [Part1], Section 2.8>
accept-ext = OWS ";" OWS token ["=" word]
accept-params = OWS ";" OWS "q=" qvalue *accept-ext
asctime-date = day-name SP date3 SP time-of-day SP year
attribute = token

charset = token
codings = content-coding / "identity" / "*"
comment = <comment, defined in [Part1], Section 3.2.4>
content-coding = token

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-1 = %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
delta-seconds = 1*DIGIT

expect-name = token
expect-param = expect-name [BWS "=" BWS expect-value]
expect-value = token / quoted-string
expectation = expect-name [BWS "=" BWS expect-value] *(OWS ";" [
OWS expect-param])

hour = 2DIGIT

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

mailbox = <mailbox, defined in [RFC5322], Section 3.4>
media-range = ("*"/*" / (type "/"*") / (type "/" subtype)) *(OWS
";" OWS parameter)

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

parameter = attribute "=" value
partial-URI = <partial-URI, defined in [Part1], Section 2.8>
product = token [ "/" product-version ]
product-version = token

quoted-string = <quoted-string, defined in [Part1], Section 3.2.4>
qvalue = <qvalue, defined in [Part1], Section 4.3.1>

rfc1123-date = day-name ", " SP date1 SP time-of-day SP GMT
rfc850-date = day-name-1 ", " SP date2 SP time-of-day SP GMT

second = 2DIGIT
subtype = token

time-of-day = hour ":" minute ":" second
token = <token, defined in [Part1], Section 3.2.4>
type = token

value = word

word = <word, defined in [Part1], Section 3.2.4>

year = 4DIGIT
```

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

F.1. Since RFC 2616

Extracted relevant partitions from [RFC2616].

F.2. Since draft-ietf-httpbis-p2-semantic-00

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/5>>: "Via is a MUST" (<http://purl.org/NET/http-errata#via-must>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/6>>: "Fragments allowed in Location" (<http://purl.org/NET/http-errata#location-fragments>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/10>>: "Safe Methods vs Redirection" (<http://purl.org/NET/http-errata#saferedirect>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/17>>: "Revise description of the POST method" (<http://purl.org/NET/http-errata#post>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/35>>: "Normative and Informative references"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/42>>: "RFC2606 Compliance"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/65>>: "Informative references"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/84>>: "Redundant cross-references"

Other changes:

- o Move definitions of 304 and 412 condition codes to [Part4]

F.3. Since draft-ietf-httpbis-p3-payload-00

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/8>>: "Media Type Registrations" (<http://purl.org/NET/http-errata#media-reg>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/14>>: "Clarification regarding quoting of charset values" (<http://purl.org/NET/http-errata#character-sets>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/16>>: "Remove 'identity' token references" (<http://purl.org/NET/http-errata#identity>)

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/25>>: "Accept-Encoding BNF"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/35>>: "Normative and Informative references"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/46>>: "RFC1700 references"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/55>>: "Updating to RFC4288"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/65>>: "Informative references"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/66>>: "ISO-8859-1 Reference"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/68>>: "Encoding References Normative"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/86>>: "Normative up-to-date references"

F.4. Since draft-ietf-httpbis-p2-semantics-01

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/21>>: "PUT side effects"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/91>>: "Duplicate Host header requirements"

Ongoing work on ABNF conversion

(<<http://tools.ietf.org/wg/httpbis/trac/ticket/36>>):

- o Move "Product Tokens" section (back) into Part 1, as "token" is used in the definition of the Upgrade header field.
- o Add explicit references to BNF syntax and rules imported from other parts of the specification.
- o Copy definition of delta-seconds from Part6 instead of referencing it.

F.5. Since draft-ietf-httpbis-p3-payload-01

Ongoing work on ABNF conversion

(<<http://tools.ietf.org/wg/httpbis/trac/ticket/36>>):

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

F.6. Since draft-ietf-httpbis-p2-semantics-02

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/24>>: "Requiring Allow in 405 responses"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/59>>: "Status Code Registry"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/61>>: "Redirection vs. Location"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/70>>: "Cacheability of 303 response"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/76>>: "305 Use Proxy"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/105>>: "Classification for Allow header field"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/112>>: "PUT - 'store under' vs 'store at'"

Ongoing work on IANA Message Header Field Registration

(<<http://tools.ietf.org/wg/httpbis/trac/ticket/40>>):

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

Ongoing work on ABNF conversion

(<<http://tools.ietf.org/wg/httpbis/trac/ticket/36>>):

- o Replace string literals when the string really is case-sensitive (method).

F.7. Since draft-ietf-httpbis-p3-payload-02

Closed issues:

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

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

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

F.8. Since draft-ietf-httpbis-p2-semantics-03

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/98>>: "OPTIONS request bodies"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/119>>: "Description of CONNECT should refer to RFC2817"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/125>>: "Location Content-Location reference request/response mixup"

Ongoing work on Method Registry
(<<http://tools.ietf.org/wg/httpbis/trac/ticket/72>>):

- o Added initial proposal for registration process, plus initial content (non-HTTP/1.1 methods to be added by a separate specification).

F.9. Since draft-ietf-httpbis-p3-payload-03

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/67>>: "Quoting Charsets"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/113>>: "language tag matching (Accept-Language) vs RFC4647"

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

Other changes:

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

F.10. Since draft-ietf-httpbis-p2-semantics-04

Closed issues:

- o <http://tools.ietf.org/wg/httpbis/trac/ticket/103>: "Content-"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/132>: "RFC 2822 is updated by RFC 5322"

Ongoing work on ABNF conversion

(<http://tools.ietf.org/wg/httpbis/trac/ticket/36>):

- o Use "/" instead of "|" for alternatives.
- o Introduce new ABNF rules for "bad" whitespace ("BWS"), optional whitespace ("OWS") and required whitespace ("RWS").
- o Rewrite ABNFs to spell out whitespace rules, factor out header field value format definitions.

F.11. Since draft-ietf-httpbis-p3-payload-04

Closed issues:

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

Ongoing work on ABNF conversion

(<http://tools.ietf.org/wg/httpbis/trac/ticket/36>):

- o Use "/" instead of "|" for alternatives.
- o Introduce new ABNF rules for "bad" whitespace ("BWS"), optional whitespace ("OWS") and required whitespace ("RWS").
- o Rewrite ABNFs to spell out whitespace rules, factor out header field value format definitions.

F.12. Since draft-ietf-httpbis-p2-semantics-05

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/94>>: "reason-phrase BNF"

Final work on ABNF conversion

(<<http://tools.ietf.org/wg/httpbis/trac/ticket/36>>):

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

F.13. Since draft-ietf-httpbis-p3-payload-05

Closed issues:

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

Final work on ABNF conversion

(<<http://tools.ietf.org/wg/httpbis/trac/ticket/36>>):

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

Other changes:

- o Move definition of quality values into Part 1.

F.14. Since draft-ietf-httpbis-p2-semantics-06

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/144>>: "Clarify when Referer is sent"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/164>>: "status codes vs methods"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/170>>: "Do not require "updates" relation for specs that register status codes or method names"

F.15. Since draft-ietf-httpbis-p3-payload-06

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/80>>: "Content-Location isn't special"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/155>>: "Content Sniffing"

F.16. Since draft-ietf-httpbis-p2-semantics-07

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/27>>: "Idempotency"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/33>>: "TRACE security considerations"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/110>>: "Clarify rules for determining what entities a response carries"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/140>>: "update note citing RFC 1945 and 2068"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/182>>: "update note about redirect limit"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/191>>: "Location header field ABNF should use 'URI'"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/192>>: "fragments in Location vs status 303"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/198>>: "move IANA registrations for optional status codes"

Partly resolved issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/171>>: "Are OPTIONS and TRACE safe?"

F.17. Since draft-ietf-httpbis-p3-payload-07

Closed issues:

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

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/110>>: "Clarify rules for determining what entities a response carries"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/154>>: "Content-Location base-setting problems"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/155>>: "Content Sniffing"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/188>>: "pick IANA policy (RFC5226) for Transfer Coding / Content Coding"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/189>>: "move definitions of gzip/deflate/compress to part 1"

Partly resolved issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/148>>: "update IANA requirements wrt Transfer-Coding values" (add the IANA Considerations subsection)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/149>>: "update IANA requirements wrt Content-Coding values" (add the IANA Considerations subsection)

F.18. Since draft-ietf-httpbis-p2-semantics-08

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/10>>: "Safe Methods vs Redirection" (we missed the introduction to the 3xx status codes when fixing this previously)

F.19. Since draft-ietf-httpbis-p3-payload-08

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/81>>: "Content Negotiation for media types"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/181>>: "Accept-Language: which RFC4647 filtering?"

F.20. Since draft-ietf-httpbis-p2-semantics-09

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/43>>: "Fragment combination / precedence during redirects"

Partly resolved issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/185>>: "Location header field payload handling"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/196>>: "Term for the requested resource's URI"

F.21. Since draft-ietf-httpbis-p3-payload-09

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/122>>: "MIME-Version not listed in P1, general header fields"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/143>>: "IANA registry for content/transfer encodings"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/155>>: "Content Sniffing"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/200>>: "use of term "word" when talking about header field structure"

Partly resolved issues:

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

F.22. Since draft-ietf-httpbis-p2-semantics-10

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/69>>: "Clarify 'Requested Variant' "
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/109>>: "Clarify entity / representation / variant terminology"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/139>>: "Methods and Caching"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/190>>: "OPTIONS vs Max-Forwards"

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/199>>: "Status codes and caching"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/220>>: "consider removing the 'changes from 2068' sections"

F.23. Since draft-ietf-httpbis-p3-payload-10

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/69>>: "Clarify 'Requested Variant' "
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/80>>: "Content-Location isn't special"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/90>>: "Delimiting messages with multipart/byteranges"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/109>>: "Clarify entity / representation / variant terminology"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/136>>: "confusing req. language for Content-Location"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/167>>: "Content-Location on 304 responses"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/183>>: "'requested resource' in content-encoding definition"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/220>>: "consider removing the 'changes from 2068' sections"

Partly resolved issues:

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

F.24. Since draft-ietf-httpbis-p2-semantics-11

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/229>>: "Considerations for new status codes"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/230>>: "Considerations for new methods"

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/232>>: "User-Agent guidelines" (relating to the 'User-Agent' header field)

F.25. Since draft-ietf-httpbis-p3-payload-11

Closed issues:

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

F.26. Since draft-ietf-httpbis-p2-semantics-12

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/43>>: "Fragment combination / precedence during redirects" (added warning about having a fragid on the redirect might cause inconvenience in some cases)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/79>>: "Content-* vs. PUT"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/88>>: "205 Bodies"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/102>>: "Understanding Content-* on non-PUT requests"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/103>>: "Content-*"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/104>>: "Header field type defaulting"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/112>>: "PUT - 'store under' vs 'store at'"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/137>>: "duplicate ABNF for reason-phrase"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/180>>: "Note special status of Content-* prefix in header field registration procedures"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/203>>: "Max-Forwards vs extension methods"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/213>>: "What is the value space of HTTP status codes?" (actually fixed in draft-ietf-httpbis-p2-semantics-11)

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/224>>: "Header Field Classification"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/225>>: "PUT side effect: invalidation or just stale?"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/226>>: "proxies not supporting certain methods"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/239>>: "Migrate CONNECT from RFC2817 to p2"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/240>>: "Migrate Upgrade details from RFC2817"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/267>>: "clarify PUT semantics"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/275>>: "duplicate ABNF for 'Method'"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/276>>: "untangle ABNFs for header fields"

F.27. Since draft-ietf-httpbis-p3-payload-12

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/224>>: "Header Field Classification"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/276>>: "untangle ABNFs for header fields"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/277>>: "potentially misleading MAY in media-type def"

F.28. Since draft-ietf-httpbis-p2-semantics-13

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/276>>: "untangle ABNFs for header fields"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/251>>: "message body in CONNECT request"

F.29. Since draft-ietf-httpbis-p3-payload-13

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/20>>: "Default charsets for text media types"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/178>>: "Content-MD5 and partial responses"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/276>>: "untangle ABNFs for header fields"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/281>>: "confusing undefined parameter in media range example"

F.30. Since draft-ietf-httpbis-p2-semantic-14

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/255>>: "Clarify status code for rate limiting"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/294>>: "clarify 403 forbidden"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/296>>: "Clarify 203 Non-Authoritative Information"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/298>>: "update default reason phrase for 413"

F.31. Since draft-ietf-httpbis-p3-payload-14

None.

F.32. Since draft-ietf-httpbis-p2-semantic-15

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/285>>: "Strength of requirements on Accept re: 406"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/303>>: "400 response isn't generic"

F.33. Since draft-ietf-httpbis-p3-payload-15

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/285>>: "Strength of requirements on Accept re: 406"

F.34. Since draft-ietf-httpbis-p2-semantics-16

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/160>>: "Redirects and non-GET methods"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/186>>: "Document HTTP's error-handling philosophy"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/231>>: "Considerations for new header fields"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/310>>: "clarify 303 redirect on HEAD"

F.35. Since draft-ietf-httpbis-p3-payload-16

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/186>>: "Document HTTP's error-handling philosophy"

F.36. Since draft-ietf-httpbis-p2-semantics-17

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/185>>: "Location header field payload handling"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/255>>: "Clarify status code for rate limiting" (change backed out because a new status code is being defined for this purpose)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/312>>: "should there be a permanent variant of 307"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/325>>: "When are Location's semantics triggered?"

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/327>>: "'expect' grammar missing OWS"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/329>>: "header field considerations: quoted-string vs use of double quotes"

F.37. Since draft-ietf-httpbis-p3-payload-17

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/323>>: "intended maturity level vs normative references"

F.38. Since draft-ietf-httpbis-p2-semantics-18

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/227>>: "Combining HEAD responses"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/238>>: "Requirements for user intervention during redirects"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/250>>: "message-body in CONNECT response"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/295>>: "Applying original fragment to 'plain' redirected URI"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/302>>: "Misplaced text on connection handling in p2"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/331>>: "clarify that 201 doesn't require Location header fields"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/332>>: "relax requirements on hypertext in 3/4/5xx error responses"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/333>>: "example for 426 response should have a payload"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/336>>: "drop indirection entries for status codes"

F.39. Since draft-ietf-httpbis-p3-payload-18

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/330>>: "is ETag a representation header field?"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/338>>: "Content-Location doesn't constrain the cardinality of representations"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/346>>: "make IANA policy definitions consistent"

F.40. Since draft-ietf-httpbis-p2-semantics-19 and draft-ietf-httpbis-p3-payload-19

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/312>>: "should there be a permanent variant of 307"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/347>>: "clarify that 201 can imply *multiple* resources were created"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/351>>: "merge P2 and P3"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/361>>: "ABNF requirements for recipients"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/364>>: "Capturing more information in the method registry"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/368>>: "note introduction of new IANA registries as normative changes"

Index

- 1
1xx Informational (status code class) 25
- 2
2xx Successful (status code class) 26
- 3
3xx Redirection (status code class) 28
- 4

	4xx Client Error (status code class)	32
5	5xx Server Error (status code class)	36
1	100 Continue (status code)	25
	100-continue (expect value)	62
	101 Switching Protocols (status code)	25
2	200 OK (status code)	26
	201 Created (status code)	26
	202 Accepted (status code)	27
	203 Non-Authoritative Information (status code)	27
	204 No Content (status code)	27
	205 Reset Content (status code)	28
3	300 Multiple Choices (status code)	29
	301 Moved Permanently (status code)	30
	302 Found (status code)	30
	303 See Other (status code)	31
	305 Use Proxy (status code)	31
	306 (Unused) (status code)	31
	307 Temporary Redirect (status code)	32
4	400 Bad Request (status code)	32
	402 Payment Required (status code)	32
	403 Forbidden (status code)	32
	404 Not Found (status code)	33
	405 Method Not Allowed (status code)	33
	406 Not Acceptable (status code)	33
	408 Request Timeout (status code)	33
	409 Conflict (status code)	34
	410 Gone (status code)	34
	411 Length Required (status code)	34
	413 Request Representation Too Large (status code)	35
	414 URI Too Long (status code)	35
	415 Unsupported Media Type (status code)	35
	417 Expectation Failed (status code)	35
	426 Upgrade Required (status code)	35
5	500 Internal Server Error (status code)	36
	501 Not Implemented (status code)	36
	502 Bad Gateway (status code)	36

503 Service Unavailable (status code) 36
504 Gateway Timeout (status code) 37
505 HTTP Version Not Supported (status code) 37

A

Accept header field 52
Accept-Charset header field 54
Accept-Encoding header field 55
Accept-Language header field 56
Allow header field 57

C

Coding Format
 compress 42
 deflate 42
 gzip 42
compress (Coding Format) 42
CONNECT method 17
content negotiation 7
Content-Encoding header field 57
Content-Language header field 58
Content-Location header field 59
Content-Transfer-Encoding header field 79
Content-Type header field 61

D

Date header field 61
deflate (Coding Format) 42
DELETE method 16

E

Expect header field 62
Expect Values
 100-continue 62

F

From header field 63

G

GET method 12
Grammar
 Accept 52
 Accept-Charset 54
 Accept-Encoding 55
 accept-ext 52
 Accept-Language 56
 accept-params 52
 Allow 57

asctime-date 40
attribute 43
charset 41
codings 55
content-coding 41
Content-Encoding 57
Content-Language 58
Content-Location 59
Content-Type 61
Date 61
date1 39
day 39
day-name 39
day-name-1 39
delta-seconds 66
Expect 62
expect-name 62
expect-param 62
expect-value 62
expectation 62
From 63
GMT 39
hour 39
HTTP-date 38
language-range 56
language-tag 44
Location 64
Max-Forwards 65
media-range 52
media-type 42
method 8
MIME-Version 78
minute 39
month 39
obs-date 39
parameter 43
product 40
product-version 40
Referer 65
Retry-After 66
rfc850-date 40
rfc1123-date 39
second 39
Server 66
subtype 42
time-of-day 39
type 42
User-Agent 67

value 43
year 39
gzip (Coding Format) 42

H

HEAD method 12
Header Fields
Accept 52
Accept-Charset 54
Accept-Encoding 55
Accept-Language 56
Allow 57
Content-Encoding 57
Content-Language 58
Content-Location 59
Content-Transfer-Encoding 79
Content-Type 61
Date 61
Expect 62
From 63
Location 63
Max-Forwards 65
MIME-Version 78
Referer 65
Retry-After 66
Server 66
User-Agent 67

I

Idempotent Methods 9

L

Location header field 63

M

Max-Forwards header field 65
Methods
CONNECT 17
DELETE 16
GET 12
HEAD 12
OPTIONS 11
POST 13
PUT 14
TRACE 16
MIME-Version header field 78

O

OPTIONS method 11

P

payload 45
POST method 13
PUT method 14

R

Referer header field 65
representation 45
Retry-After header field 66

S

Safe Methods 9
selected representation 47
Server header field 66
Status Codes
 100 Continue 25
 101 Switching Protocols 25
 200 OK 26
 201 Created 26
 202 Accepted 27
 203 Non-Authoritative Information 27
 204 No Content 27
 205 Reset Content 28
 300 Multiple Choices 29
 301 Moved Permanently 30
 302 Found 30
 303 See Other 31
 305 Use Proxy 31
 306 (Unused) 31
 307 Temporary Redirect 32
 400 Bad Request 32
 402 Payment Required 32
 403 Forbidden 32
 404 Not Found 33
 405 Method Not Allowed 33
 406 Not Acceptable 33
 408 Request Timeout 33
 409 Conflict 34
 410 Gone 34
 411 Length Required 34
 413 Request Representation Too Large 35
 414 URI Too Long 35
 415 Unsupported Media Type 35
 417 Expectation Failed 35
 426 Upgrade Required 35
 500 Internal Server Error 36

501 Not Implemented	36
502 Bad Gateway	36
503 Service Unavailable	36
504 Gateway Timeout	37
505 HTTP Version Not Supported	37
Status Codes Classes	
1xx Informational	25
2xx Successful	26
3xx Redirection	28
4xx Client Error	32
5xx Server Error	36

T

TRACE method	16
--------------	----

U

User-Agent header field	67
-------------------------	----

Authors' Addresses

Roy T. Fielding (editor)
Adobe Systems Incorporated
345 Park Ave
San Jose, CA 95110
USA

EMail: fielding@gbiv.com
URI: <http://roy.gbiv.com/>

Yves Lafon (editor)
World Wide Web Consortium
W3C / ERCIM
2004, rte des Lucioles
Sophia-Antipolis, AM 06902
France

EMail: ylafon@w3.org
URI: <http://www.raubacapeu.net/people/yves/>

Julian F. Reschke (editor)
greenbytes GmbH
Hafenweg 16
Muenster, NW 48155
Germany

EMail: julian.reschke@greenbytes.de
URI: <http://greenbytes.de/tech/webdav/>

HTTPbis Working Group
Internet-Draft
Obsoletes: 2616 (if approved)
Intended status: Standards Track
Expires: January 17, 2013

R. Fielding, Ed.
Adobe
Y. Lafon, Ed.
W3C
J. Reschke, Ed.
greenbytes
July 16, 2012

HTTP/1.1, part 4: Conditional Requests
draft-ietf-httpbis-p4-conditional-20

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypertext information systems. This document defines HTTP/1.1 conditional requests, including metadata header fields for indicating state changes, request header fields for making preconditions on such state, and rules for constructing the responses to a conditional request when one or more preconditions evaluate to false.

Editorial Note (To be removed by RFC Editor)

Discussion of this draft takes place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org), which is archived at <http://lists.w3.org/Archives/Public/ietf-http-wg/>.

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 D.1.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference

material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 17, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	4
1.1. Conformance and Error Handling	4
1.2. Syntax Notation	5
2. Validators	5
2.1. Weak versus Strong	6
2.2. Last-Modified	7
2.2.1. Generation	8
2.2.2. Comparison	8
2.3. ETag	9
2.3.1. Generation	10
2.3.2. Comparison	11
2.3.3. Example: Entity-tags varying on Content-Negotiated Resources	11
2.4. Rules for When to Use Entity-tags and Last-Modified Dates	12
3. Precondition Header Fields	14
3.1. If-Match	14
3.2. If-None-Match	15
3.3. If-Modified-Since	16
3.4. If-Unmodified-Since	17
3.5. If-Range	18
4. Status Code Definitions	18
4.1. 304 Not Modified	18
4.2. 412 Precondition Failed	19
5. Precedence	19
6. IANA Considerations	20
6.1. Status Code Registration	20
6.2. Header Field Registration	21
7. Security Considerations	21
8. Acknowledgments	21
9. References	22
9.1. Normative References	22
9.2. Informative References	22
Appendix A. Changes from RFC 2616	22
Appendix B. Imported ABNF	23
Appendix C. Collected ABNF	23
Appendix D. Change Log (to be removed by RFC Editor before publication)	24
D.1. Since draft-ietf-httpbis-p4-conditional-19	24
Index	24

1. Introduction

Conditional requests are HTTP requests [Part2] that include one or more header fields indicating a precondition to be tested before applying the method semantics to the target resource. Each precondition is based on metadata that is expected to change if the selected representation of the target resource is changed. This document defines the HTTP/1.1 conditional request mechanisms in terms of the architecture, syntax notation, and conformance criteria defined in [Part1].

Conditional GET requests are the most efficient mechanism for HTTP cache updates [Part6]. Conditionals can also be applied to state-changing methods, such as PUT and DELETE, to prevent the "lost update" problem: one client accidentally overwriting the work of another client that has been acting in parallel.

Conditional request preconditions are based on the state of the target resource as a whole (its current value set) or the state as observed in a previously obtained representation (one value in that set). A resource might have multiple current representations, each with its own observable state. The conditional request mechanisms assume that the mapping of requests to corresponding representations will be consistent over time if the server intends to take advantage of conditionals. Regardless, if the mapping is inconsistent and the server is unable to select the appropriate representation, then no harm will result when the precondition evaluates to false.

We use the term "selected representation" to refer to the current representation of the target resource that would have been selected in a successful response if the same request had used the method GET and had excluded all of the conditional request header fields. The conditional request preconditions are evaluated by comparing the values provided in the request header fields to the current metadata for the selected representation.

1.1. Conformance and Error Handling

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].

This specification targets conformance criteria according to the role of a participant in HTTP communication. Hence, HTTP requirements are placed on senders, recipients, clients, servers, user agents, intermediaries, origin servers, proxies, gateways, or caches, depending on what behavior is being constrained by the requirement. See Section 2 of [Part1] for definitions of these terms.

The verb "generate" is used instead of "send" where a requirement differentiates between creating a protocol element and merely forwarding a received element downstream.

An implementation is considered conformant if it complies with all of the requirements associated with the roles it partakes in HTTP. Note that SHOULD-level requirements are relevant here, unless one of the documented exceptions is applicable.

This document also uses ABNF to define valid protocol elements (Section 1.2). In addition to the prose requirements placed upon them, senders MUST NOT generate protocol elements that do not match the grammar defined by the ABNF rules for those protocol elements that are applicable to the sender's role. If a received protocol element is processed, the recipient MUST be able to parse any value that would match the ABNF rules for that protocol element, excluding only those rules not applicable to the recipient's role.

Unless noted otherwise, a recipient MAY attempt to recover a usable protocol element from an invalid construct. HTTP does not define specific error handling mechanisms except when they have a direct impact on security, since different applications of the protocol require different error handling strategies. For example, a Web browser might wish to transparently recover from a response where the Location header field doesn't parse according to the ABNF, whereas a systems control client might consider any form of error recovery to be dangerous.

1.2. Syntax Notation

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] with the list rule extension defined in Section 1.2 of [Part1]. Appendix B describes rules imported from other documents. Appendix C shows the collected ABNF with the list rule expanded.

2. Validators

This specification defines two forms of metadata that are commonly used to observe resource state and test for preconditions: modification dates (Section 2.2) and opaque entity tags (Section 2.3). Additional metadata that reflects resource state has been defined by various extensions of HTTP, such as WebDAV [RFC4918], that are beyond the scope of this specification. A resource metadata value is referred to as a "validator" when it is used within a precondition.

2.1. Weak versus Strong

Validators come in two flavors: strong or weak. Weak validators are easy to generate but are far less useful for comparisons. Strong validators are ideal for comparisons but can be very difficult (and occasionally impossible) to generate efficiently. Rather than impose that all forms of resource adhere to the same strength of validator, HTTP exposes the type of validator in use and imposes restrictions on when weak validators can be used as preconditions.

A "strong validator" is a representation metadata value that **MUST** be changed to a new, previously unused or guaranteed unique, value whenever a change occurs to the representation data such that a change would be observable in the payload body of a 200 (OK) response to GET.

A strong validator **MAY** be changed for other reasons, such as when a semantically significant part of the representation metadata is changed (e.g., Content-Type), but it is in the best interests of the origin server to only change the value when it is necessary to invalidate the stored responses held by remote caches and authoring tools. A strong validator **MUST** be unique across all representations of a given resource, such that no two representations of that resource share the same validator unless their payload body would be identical.

Cache entries might persist for arbitrarily long periods, regardless of expiration times. Thus, a cache might attempt to validate an entry using a validator that it obtained in the distant past. A strong validator **MUST** be unique across all versions of all representations associated with a particular resource over time. However, there is no implication of uniqueness across representations of different resources (i.e., the same strong validator might be in use for representations of multiple resources at the same time and does not imply that those representations are equivalent).

There are a variety of strong validators used in practice. The best are based on strict revision control, wherein each change to a representation always results in a unique node name and revision identifier being assigned before the representation is made accessible to GET. A collision-resistant hash function applied to the representation data is also sufficient if the data is available prior to the response header fields being sent and the digest does not need to be recalculated every time a validation request is received. However, if a resource has distinct representations that differ only in their metadata, such as might occur with content negotiation over media types that happen to share the same data format, then the origin server **SHOULD** incorporate additional

information in the validator to distinguish those representations and avoid confusing cache behavior.

In contrast, a "weak validator" is a representation metadata value that might not be changed for every change to the representation data. This weakness might be due to limitations in how the value is calculated, such as clock resolution or an inability to ensure uniqueness for all possible representations of the resource, or due to a desire by the resource owner to group representations by some self-determined set of equivalency rather than unique sequences of data. An origin server **SHOULD** change a weak entity-tag whenever it considers prior representations to be unacceptable as a substitute for the current representation. In other words, a weak entity-tag ought to change whenever the origin server wants caches to invalidate old responses.

For example, the representation of a weather report that changes in content every second, based on dynamic measurements, might be grouped into sets of equivalent representations (from the origin server's perspective) with the same weak validator in order to allow cached representations to be valid for a reasonable period of time (perhaps adjusted dynamically based on server load or weather quality). Likewise, a representation's modification time, if defined with only one-second resolution, might be a weak validator if it is possible for the representation to be modified twice during a single second and retrieved between those modifications.

A "use" of a validator occurs when either a client generates a request and includes the validator in a precondition or when a server compares two validators. Weak validators are only usable in contexts that do not depend on exact equality of a representation's payload body. Strong validators are usable and preferred for all conditional requests, including cache validation, partial content ranges, and "lost update" avoidance.

2.2. Last-Modified

The "Last-Modified" header field indicates the date and time at which the origin server believes the selected representation was last modified.

Last-Modified = HTTP-date

An example of its use is

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

2.2.1. Generation

Origin servers SHOULD send Last-Modified for any selected representation for which a last modification date can be reasonably and consistently determined, since its use in conditional requests and evaluating cache freshness ([Part6]) results in a substantial reduction of HTTP traffic on the Internet and can be a significant factor in improving service scalability and reliability.

A representation is typically the sum of many parts behind the resource interface. The last-modified time would usually be the most recent time that any of those parts were changed. How that value is determined for any given resource is an implementation detail beyond the scope of this specification. What matters to HTTP is how recipients of the Last-Modified header field can use its value to make conditional requests and test the validity of locally cached responses.

An origin server SHOULD obtain the Last-Modified value of the representation as close as possible to the time that it generates the Date field value for its response. This allows a recipient to make an accurate assessment of the representation's modification time, especially if the representation changes near the time that the response is generated.

An origin server with a clock MUST NOT send a Last-Modified date that is later than the server's time of message origination (Date). If the last modification time is derived from implementation-specific metadata that evaluates to some time in the future, according to the origin server's clock, then the origin server MUST replace that value with the message origination date. This prevents a future modification date from having an adverse impact on cache validation.

An origin server without a clock MUST NOT assign Last-Modified values to a response unless these values were associated with the resource by some other system or user with a reliable clock.

2.2.2. Comparison

A Last-Modified time, when used as a validator in a request, is implicitly weak unless it is possible to deduce that it is strong, using the following rules:

- o The validator is being compared by an origin server to the actual current validator for the representation and,
- o That origin server reliably knows that the associated representation did not change twice during the second covered by

the presented validator.

or

- o The validator is about to be used by a client in an If-Modified-Since, If-Unmodified-Since header field, because the client has a cache entry, or If-Range for the associated representation, and
- o That cache entry includes a Date value, which gives the time when the origin server sent the original response, and
- o The presented Last-Modified time is at least 60 seconds before the Date value.

or

- o The validator is being compared by an intermediate cache to the validator stored in its cache entry for the representation, and
- o That cache entry includes a Date value, which gives the time when the origin server sent the original response, and
- o The presented Last-Modified time is at least 60 seconds before the Date value.

This method relies on the fact that if two different responses were sent by the origin server during the same second, but both had the same Last-Modified time, then at least one of those responses would have a Date value equal to its Last-Modified time. The arbitrary 60-second limit guards against the possibility that the Date and Last-Modified values are generated from different clocks, or at somewhat different times during the preparation of the response. An implementation MAY use a value larger than 60 seconds, if it is believed that 60 seconds is too short.

2.3. ETag

The "ETag" header field provides the current entity-tag for the selected representation. An entity-tag is an opaque validator for differentiating between multiple representations of the same resource, regardless of whether those multiple representations are due to resource state changes over time, content negotiation resulting in multiple representations being valid at the same time, or both. An entity-tag consists of an opaque quoted string, possibly prefixed by a weakness indicator.

```
ETag          = entity-tag

entity-tag    = [ weak ] opaque-tag
weak          = %x57.2F ; "W/", case-sensitive
opaque-tag    = DQUOTE *etagc DQUOTE
etagc        = %x21 / %x23-7E / obs-text
              ; VCHAR except double quotes, plus obs-text
```

Note: Previously, opaque-tag was defined to be a quoted-string ([RFC2616], Section 3.11), thus some recipients might perform backslash unescaping. Servers therefore ought to avoid backslash characters in entity tags.

An entity-tag can be more reliable for validation than a modification date in situations where it is inconvenient to store modification dates, where the one-second resolution of HTTP date values is not sufficient, or where modification dates are not consistently maintained.

Examples:

```
ETag: "xyzzzy"
ETag: W/"xyzzzy"
ETag: ""
```

An entity-tag can be either a weak or strong validator, with strong being the default. If an origin server provides an entity-tag for a representation and the generation of that entity-tag does not satisfy the requirements for a strong validator (Section 2.1), then that entity-tag **MUST** be marked as weak by prefixing its opaque value with "W/" (case-sensitive).

2.3.1. Generation

The principle behind entity-tags is that only the service author knows the implementation of a resource well enough to select the most accurate and efficient validation mechanism for that resource, and that any such mechanism can be mapped to a simple sequence of octets for easy comparison. Since the value is opaque, there is no need for the client to be aware of how each entity-tag is constructed.

For example, a resource that has implementation-specific versioning applied to all changes might use an internal revision number, perhaps combined with a variance identifier for content negotiation, to accurately differentiate between representations. Other implementations might use a collision-resistant hash of representation content, a combination of various filesystem attributes, or a modification timestamp that has sub-second

resolution.

Origin servers SHOULD send ETag for any selected representation for which detection of changes can be reasonably and consistently determined, since the entity-tag's use in conditional requests and evaluating cache freshness ([Part6]) can result in a substantial reduction of HTTP network traffic and can be a significant factor in improving service scalability and reliability.

2.3.2. Comparison

There are two entity-tag comparison functions, depending on whether the comparison context allows the use of weak validators or not:

- o The strong comparison function: in order to be considered equal, both opaque-tags MUST be identical character-by-character, and both MUST NOT be weak.
- o The weak comparison function: in order to be considered equal, both opaque-tags MUST be identical character-by-character, but either or both of them MAY be tagged as "weak" without affecting the result.

The example below shows the results for a set of entity-tag pairs, and both the weak and strong comparison function results:

ETag 1	ETag 2	Strong Comparison	Weak Comparison
W/"1"	W/"1"	no match	match
W/"1"	W/"2"	no match	no match
W/"1"	"1"	no match	match
"1"	"1"	match	match

2.3.3. Example: Entity-tags varying on Content-Negotiated Resources

Consider a resource that is subject to content negotiation (Section 8 of [Part2]), and where the representations returned upon a GET request vary based on the Accept-Encoding request header field (Section 9.3 of [Part2]):

>> Request:

```
GET /index HTTP/1.1
Host: www.example.com
Accept-Encoding: gzip
```

In this case, the response might or might not use the gzip content coding. If it does not, the response might look like:

>> Response:

```
HTTP/1.1 200 OK
Date: Thu, 26 Mar 2010 00:05:00 GMT
ETag: "123-a"
Content-Length: 70
Vary: Accept-Encoding
Content-Type: text/plain
```

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

An alternative representation that does use gzip content coding would be:

>> Response:

```
HTTP/1.1 200 OK
Date: Thu, 26 Mar 2010 00:05:00 GMT
ETag: "123-b"
Content-Length: 43
Vary: Accept-Encoding
Content-Type: text/plain
Content-Encoding: gzip
```

...binary data...

Note: Content codings are a property of the representation, so therefore an entity-tag of an encoded representation has to be distinct from an unencoded representation to prevent conflicts during cache updates and range requests. In contrast, transfer codings (Section 4 of [Part1]) apply only during message transfer and do not require distinct entity-tags.

2.4. Rules for When to Use Entity-tags and Last-Modified Dates

We adopt a set of rules and recommendations for origin servers, clients, and caches regarding when various validator types ought to be used, and for what purposes.

HTTP/1.1 origin servers:

- o SHOULD send an entity-tag validator unless it is not feasible to generate one.
- o MAY send a weak entity-tag instead of a strong entity-tag, if performance considerations support the use of weak entity-tags, or if it is unfeasible to send a strong entity-tag.
- o SHOULD send a Last-Modified value if it is feasible to send one.

In other words, the preferred behavior for an HTTP/1.1 origin server is to send both a strong entity-tag and a Last-Modified value.

HTTP/1.1 clients:

- o MUST use that entity-tag in any cache-conditional request (using If-Match or If-None-Match) if an entity-tag has been provided by the origin server.
- o SHOULD use the Last-Modified value in non-subrange cache-conditional requests (using If-Modified-Since) if only a Last-Modified value has been provided by the origin server.
- o MAY use the Last-Modified value in subrange cache-conditional requests (using If-Unmodified-Since) if only a Last-Modified value has been provided by an HTTP/1.0 origin server. The user agent SHOULD provide a way to disable this, in case of difficulty.
- o SHOULD use both validators in cache-conditional requests if both an entity-tag and a Last-Modified value have been provided by the origin server. This allows both HTTP/1.0 and HTTP/1.1 caches to respond appropriately.

An HTTP/1.1 origin server, upon receiving a conditional request that includes both a Last-Modified date (e.g., in an If-Modified-Since or If-Unmodified-Since header field) and one or more entity-tags (e.g., in an If-Match, If-None-Match, or If-Range header field) as cache validators, MUST NOT return a response status code of 304 (Not Modified) unless doing so is consistent with all of the conditional header fields in the request.

An HTTP/1.1 caching proxy, upon receiving a conditional request that includes both a Last-Modified date and one or more entity-tags as cache validators, MUST NOT return a locally cached response to the client unless that cached response is consistent with all of the conditional header fields in the request.

Note: The general principle behind these rules is that HTTP/1.1 servers and clients ought to transmit as much non-redundant

information as is available in their responses and requests. HTTP/1.1 systems receiving this information will make the most conservative assumptions about the validators they receive.

HTTP/1.0 clients and caches might ignore entity-tags. Generally, last-modified values received or used by these systems will support transparent and efficient caching, and so HTTP/1.1 origin servers still ought to provide Last-Modified values.

3. Precondition Header Fields

This section defines the syntax and semantics of HTTP/1.1 header fields for applying preconditions on requests. Section 5 defines the order of evaluation when more than one precondition is present in a request.

3.1. If-Match

The "If-Match" header field can be used to make a request method conditional on the current existence or value of an entity-tag for one or more representations of the target resource.

If-Match is generally useful for resource update requests, such as PUT requests, as a means for protecting against accidental overwrites when multiple clients are acting in parallel on the same resource (i.e., the "lost update" problem). An If-Match field-value of "*" places the precondition on the existence of any current representation for the target resource.

If-Match = "*" / 1#entity-tag

The If-Match condition is met if and only if any of the entity-tags listed in the If-Match field value match the entity-tag of the selected representation for the target resource (as per Section 2.3.2), or if "*" is given and any current representation exists for the target resource.

If the condition is met, the server MAY perform the request method as if the If-Match header field was not present.

Origin servers MUST NOT perform the requested method if the condition is not met; instead they MUST respond with the 412 (Precondition Failed) status code.

Proxy servers using a cached response as the selected representation MUST NOT perform the requested method if the condition is not met; instead, they MUST forward the request towards the origin server.

If the request would, without the If-Match header field, result in anything other than a 2xx (Successful) or 412 (Precondition Failed) status code, then the If-Match header field MUST be ignored.

Examples:

```
If-Match: "xyzzy"
If-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
If-Match: *
```

3.2. If-None-Match

The "If-None-Match" header field can be used to make a request method conditional on not matching any of the current entity-tag values for representations of the target resource.

If-None-Match is primarily used in conditional GET requests to enable efficient updates of cached information with a minimum amount of transaction overhead. A client that has one or more representations previously obtained from the target resource can send If-None-Match with a list of the associated entity-tags in the hope of receiving a 304 (Not Modified) response if at least one of those representations matches the selected representation.

If-None-Match can also be used with a value of "*" to prevent an unsafe request method (e.g., PUT) from inadvertently modifying an existing representation of the target resource when the client believes that the resource does not have a current representation. This is a variation on the "lost update" problem that might arise if more than one client attempts to create an initial representation for the target resource.

```
If-None-Match = "*" / 1#entity-tag
```

The If-None-Match condition is met if and only if none of the entity-tags listed in the If-None-Match field value match the entity-tag of the selected representation for the target resource (as per Section 2.3.2), or if "*" is given and no current representation exists for that resource.

If the condition is not met, the server MUST NOT perform the requested method. Instead, if the request method was GET or HEAD, the server SHOULD respond with a 304 (Not Modified) status code, including the cache-related header fields (particularly ETag) of the selected representation that has a matching entity-tag. For all other request methods, the server MUST respond with a 412 (Precondition Failed) status code.

If the condition is met, the server MAY perform the requested method as if the If-None-Match header field did not exist, but MUST also ignore any If-Modified-Since header field(s) in the request. That is, if no entity-tags match, then the server MUST NOT return a 304 (Not Modified) response.

If the request would, without the If-None-Match header field, result in anything other than a 2xx (Successful) or 304 (Not Modified) status code, then the If-None-Match header field MUST be ignored. (See Section 2.4 for a discussion of server behavior when both If-Modified-Since and If-None-Match appear in the same request.)

Examples:

```
If-None-Match: "xyzzy"
If-None-Match: W/"xyzzy"
If-None-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
If-None-Match: W/"xyzzy", W/"r2d2xxxx", W/"c3piozzzz"
If-None-Match: *
```

3.3. If-Modified-Since

The "If-Modified-Since" header field can be used with GET or HEAD to make the method conditional by modification date: if the selected representation has not been modified since the time specified in this field, then do not perform the request method; instead, respond as detailed below.

If-Modified-Since = HTTP-date

An example of the field is:

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

A GET method with an If-Modified-Since header field and no Range header field requests that the selected representation be transferred only if it has been modified since the date given by the If-Modified-Since header field. The algorithm for determining this includes the following cases:

1. If the request would normally result in anything other than a 200 (OK) status code, or if the passed If-Modified-Since date is invalid, the response is exactly the same as for a normal GET. A date which is later than the server's current time is invalid.
2. If the selected representation has been modified since the If-Modified-Since date, the response is exactly the same as for a normal GET.

3. If the selected representation has not been modified since a valid If-Modified-Since date, the server SHOULD return a 304 (Not Modified) response.

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

Note: The Range header field modifies the meaning of If-Modified-Since; see Section 5.4 of [Part5] for full details.

Note: If-Modified-Since times are interpreted by the server, whose clock might not be synchronized with the client.

Note: When handling an If-Modified-Since header field, some servers will use an exact date comparison function, rather than a less-than function, for deciding whether to send a 304 (Not Modified) response. To get best results when sending an If-Modified-Since header field for cache validation, clients are advised to use the exact date string received in a previous Last-Modified header field whenever possible.

Note: If a client uses an arbitrary date in the If-Modified-Since header field instead of a date taken from the Last-Modified header field for the same request, the client needs to be aware that this date is interpreted in the server's understanding of time. Unsynchronized clocks and rounding problems, due to the different encodings of time between the client and server, are concerns. This includes the possibility of race conditions if the document has changed between the time it was first requested and the If-Modified-Since date of a subsequent request, and the possibility of clock-skew-related problems if the If-Modified-Since date is derived from the client's clock without correction to the server's clock. Corrections for different time bases between client and server are at best approximate due to network latency.

3.4. If-Unmodified-Since

The "If-Unmodified-Since" header field can be used to make a request method conditional by modification date: if the selected representation has been modified since the time specified in this field, then the server MUST NOT perform the requested operation and MUST instead respond with the 412 (Precondition Failed) status code. If the selected representation has not been modified since the time specified in this field, the server SHOULD perform the request method as if the If-Unmodified-Since header field were not present.

If-Unmodified-Since = HTTP-date

An example of the field is:

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

If a request normally (i.e., in absence of the If-Unmodified-Since header field) would result in anything other than a 2xx (Successful) or 412 (Precondition Failed) status code, the If-Unmodified-Since header field SHOULD be ignored.

If the specified date is invalid, the header field MUST be ignored.

3.5. If-Range

The "If-Range" header field provides a special conditional request mechanism that is similar to If-Match and If-Unmodified-Since but specific to HTTP range requests. If-Range is defined in Section 5.3 of [Part5].

4. Status Code Definitions

4.1. 304 Not Modified

The 304 status code indicates that a conditional GET request has been received and would have resulted in a 200 (OK) response if it were not for the fact that the condition has evaluated to false. In other words, there is no need for the server to transfer a representation of the target resource because the client's request indicates that it already has a valid representation, as indicated by the 304 response header fields, and is therefore redirecting the client to make use of that stored representation as if it were the payload of a 200 response. The 304 response MUST NOT contain a message-body, and thus is always terminated by the first empty line after the header fields.

A 304 response MUST include a Date header field (Section 9.10 of [Part2]) unless the origin server does not have a clock that can provide a reasonable approximation of the current time. If a 200 (OK) response to the same request would have included any of the header fields Cache-Control, Content-Location, ETag, Expires, or Vary, then those same header fields MUST be sent in a 304 response.

Since the goal of a 304 response is to minimize information transfer when the recipient already has one or more cached representations, the response SHOULD NOT include representation metadata other than the above listed fields unless said metadata exists for the purpose of guiding cache updates (e.g., future HTTP extensions).

If the recipient of a 304 response does not have a cached representation corresponding to the entity-tag indicated by the 304

response, then the recipient **MUST NOT** use the 304 to update its own cache. If this conditional request originated with an outbound client, such as a user agent with its own cache sending a conditional GET to a shared proxy, then the 304 response **MAY** be forwarded to that client. Otherwise, the recipient **MUST** disregard the 304 response and repeat the request without any preconditions.

If a cache uses a received 304 response to update a cache entry, the cache **MUST** update the entry to reflect any new field values given in the response.

4.2. 412 Precondition Failed

The 412 status code indicates that one or more preconditions given in the request header fields evaluated to false when tested on the server. This response code allows the client to place preconditions on the current resource state (its current representations and metadata) and thus prevent the request method from being applied if the target resource is in an unexpected state.

5. Precedence

When more than one conditional request header field is present in a request, the order in which the fields are evaluated becomes important. In practice, the fields defined in this document are consistently implemented in a single, logical order, due to the fact that entity tags are presumed to be more accurate than date validators. For example, the only reason to send both If-Modified-Since and If-None-Match in the same GET request is to support intermediary caches that might not have implemented If-None-Match, so it makes sense to ignore the If-Modified-Since when entity tags are understood and available for the selected representation.

The general rule of conditional precedence is that exact match conditions are evaluated before cache-validating conditions and, within that order, last-modified conditions are only evaluated if the corresponding entity tag condition is not present (or not applicable because the selected representation does not have an entity tag).

Specifically, the fields defined by this specification are evaluated as follows:

1. When If-Match is present, evaluate it:
 - * if true, continue to step 3
 - * if false, respond 412 (Precondition Failed)

2. When If-Match is not present and If-Unmodified-Since is present, evaluate it:
 - * if true, continue to step 3
 - * if false, respond 412 (Precondition Failed)
3. When the method is GET and both Range and If-Range are present, evaluate it:
 - * if the validator matches, respond 206 (Partial Content)
 - * if the validator does not match, respond 200 (OK)
4. When If-None-Match is present, evaluate it:
 - * if true, all conditions are met
 - * if false for GET/HEAD, respond 304 (Not Modified)
 - * if false for other methods, respond 412 (Precondition Failed)
5. When the method is GET or HEAD, If-None-Match is not present, and If-Modified-Since is present, evaluate it:
 - * if true, all conditions are met
 - * if false, respond 304 (Not Modified)

Any extension to HTTP/1.1 that defines additional conditional request header fields ought to define its own expectations regarding the order for evaluating such fields in relation to those defined in this document and other conditionals that might be found in practice.

6. IANA Considerations

6.1. Status Code Registration

The HTTP Status Code Registry located at <http://www.iana.org/assignments/http-status-codes> shall be updated with the registrations below:

Value	Description	Reference
304	Not Modified	Section 4.1
412	Precondition Failed	Section 4.2

6.2. 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
ETag	http	standard	Section 2.3
If-Match	http	standard	Section 3.1
If-Modified-Since	http	standard	Section 3.3
If-None-Match	http	standard	Section 3.2
If-Unmodified-Since	http	standard	Section 3.4
Last-Modified	http	standard	Section 2.2

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

7. Security Considerations

No additional security considerations have been identified beyond those applicable to HTTP in general [Part1].

The validators defined by this specification are not intended to ensure the validity of a representation, guard against malicious changes, or detect man-in-the-middle attacks. At best, they enable more efficient cache updates and optimistic concurrent writes when all participants are behaving nicely. At worst, the conditions will fail and the client will receive a response that is no more harmful than an HTTP exchange without conditional requests.

8. Acknowledgments

See Section 9 of [Part1].

9. References

9.1. Normative References

- [Part1] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: Message Routing and Syntax", draft-ietf-httpbis-p1-messaging-20 (work in progress), July 2012.
- [Part2] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 2: Semantics and Payloads", draft-ietf-httpbis-p2-semantics-20 (work in progress), July 2012.
- [Part5] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 5: Range Requests", draft-ietf-httpbis-p5-range-20 (work in progress), July 2012.
- [Part6] Fielding, R., Ed., Lafon, Y., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1, part 6: Caching", draft-ietf-httpbis-p6-cache-20 (work in progress), July 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

9.2. Informative References

- [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.
- [RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.
- [RFC4918] Dusseault, L., Ed., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", RFC 4918, June 2007.

Appendix A. Changes from RFC 2616

Allow weak entity-tags in all requests except range requests (Sections 2.1 and 3.2).

Change ETag header field ABNF not to use quoted-string, thus avoiding escaping issues. (Section 2.3)

Change ABNF productions for header fields to only define the field value. (Section 3)

Appendix B. Imported ABNF

The following core rules are included by reference, as defined in Appendix B.1 of [RFC5234]: 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), and VCHAR (any visible US-ASCII character).

The rules below are defined in [Part1]:

```
OWS          = <OWS, defined in [Part1], Section 3.2.1>
obs-text     = <obs-text, defined in [Part1], Section 3.2.4>
```

The rules below are defined in other parts:

```
HTTP-date    = <HTTP-date, defined in [Part2], Section 5.1>
```

Appendix C. Collected ABNF

```
ETag = entity-tag
```

```
HTTP-date = <HTTP-date, defined in [Part2], Section 5.1>
```

```
If-Match = "*" / ( *( "," OWS ) entity-tag *( OWS "," [ OWS
entity-tag ] ) )
```

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

```
If-None-Match = "*" / ( *( "," OWS ) entity-tag *( OWS "," [ OWS
entity-tag ] ) )
```

```
If-Unmodified-Since = HTTP-date
```

```
Last-Modified = HTTP-date
```

```
OWS = <OWS, defined in [Part1], Section 3.2.1>
```

```
entity-tag = [ weak ] opaque-tag
```

```
etagc = "!" / %x23-7E ; '#' '-' '~'
       / obs-text
```

```
obs-text = <obs-text, defined in [Part1], Section 3.2.4>
```

```
opaque-tag = DQUOTE *etagc DQUOTE
```

```
weak = %x57.2F ; W/
```

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

Changes up to the first Working Group Last Call draft are summarized in <http://tools.ietf.org/html/draft-ietf-httpbis-p4-conditional-19#appendix-C>.

D.1. Since draft-ietf-httpbis-p4-conditional-19

Closed issues:

- o <http://tools.ietf.org/wg/httpbis/trac/ticket/241>: "Need to clarify eval order/interaction of conditional headers"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/354>: "ETags and Conditional Requests"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/361>: "ABNF requirements for recipients"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/363>: "Rare cases"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/365>: "Conditional Request Security Considerations"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/371>: "If-Modified-Since lacks definition for method != GET"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/372>: "refactor conditional header field descriptions"

Index

3	304 Not Modified (status code)	18
4	412 Precondition Failed (status code)	19
E	ETag header field	9
G	Grammar	
	entity-tag	10
	ETag	10
	etagc	10
	If-Match	14
	If-Modified-Since	16

	If-None-Match	15
	If-Unmodified-Since	17
	Last-Modified	7
	opaque-tag	10
	weak	10
H	Header Fields	
	ETag	9
	If-Match	14
	If-Modified-Since	16
	If-None-Match	15
	If-Unmodified-Since	17
	Last-Modified	7
I		
	If-Match header field	14
	If-Modified-Since header field	16
	If-None-Match header field	15
	If-Unmodified-Since header field	17
L		
	Last-Modified header field	7
M		
	metadata	5
S		
	selected representation	4
	Status Codes	
	304 Not Modified	18
	412 Precondition Failed	19
V		
	validator	5
	strong	6
	weak	6

Authors' Addresses

Roy T. Fielding (editor)
Adobe Systems Incorporated
345 Park Ave
San Jose, CA 95110
USA

EMail: fielding@gbiv.com
URI: <http://roy.gbiv.com/>

Yves Lafon (editor)
World Wide Web Consortium
W3C / ERCIM
2004, rte des Lucioles
Sophia-Antipolis, AM 06902
France

EMail: ylafon@w3.org
URI: <http://www.raubacapeu.net/people/yves/>

Julian F. Reschke (editor)
greenbytes GmbH
Hafenweg 16
Muenster, NW 48155
Germany

EMail: julian.reschke@greenbytes.de
URI: <http://greenbytes.de/tech/webdav/>

HTTPbis Working Group
Internet-Draft
Obsoletes: 2616 (if approved)
Intended status: Standards Track
Expires: January 17, 2013

R. Fielding, Ed.
Adobe
Y. Lafon, Ed.
W3C
J. Reschke, Ed.
greenbytes
July 16, 2012

HTTP/1.1, part 5: Range Requests
draft-ietf-httpbis-p5-range-20

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypertext information systems. This document defines range requests and the rules for constructing and combining responses to those requests.

Editorial Note (To be removed by RFC Editor)

Discussion of this draft takes place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org), which is archived at <http://lists.w3.org/Archives/Public/ietf-http-wg/>.

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

The changes in this draft are summarized in Appendix E.1.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 17, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	4
1.1. Conformance and Error Handling	4
1.2. Syntax Notation	5
2. Range Units	5
2.1. Range Specifier Registry	6
3. Status Code Definitions	6
3.1. 206 Partial Content	6
3.2. 416 Requested Range Not Satisfiable	7
4. Responses to a Range Request	7
4.1. Response to a Single and Multiple Ranges Request	7
4.2. Combining Ranges	8
5. Header Field Definitions	9
5.1. Accept-Ranges	9
5.2. Content-Range	10
5.3. If-Range	11
5.4. Range	12
5.4.1. Byte Ranges	12
5.4.2. Range Retrieval Requests	14
6. IANA Considerations	15
6.1. Status Code Registration	15
6.2. Header Field Registration	15
6.3. Range Specifier Registration	16
7. Security Considerations	16
7.1. Overlapping Ranges	16
8. Acknowledgments	16
9. References	16
9.1. Normative References	16
9.2. Informative References	17
Appendix A. Internet Media Type multipart/byteranges	18
Appendix B. Changes from RFC 2616	20
Appendix C. Imported ABNF	20
Appendix D. Collected ABNF	21
Appendix E. Change Log (to be removed by RFC Editor before publication)	22
E.1. Since draft-ietf-httpbis-p5-range-19	22
Index	22

1. Introduction

HTTP clients often encounter interrupted data transfers as a result of canceled requests or dropped connections. When a client has stored a partial representation, it is desirable to request the remainder of that representation in a subsequent request rather than transfer the entire representation. There are also a number of Web applications that benefit from being able to request only a subset of a larger representation, such as a single page of a very large document or only part of an image to be rendered by a device with limited local storage.

This document defines HTTP/1.1 range requests, partial responses, and the multipart/byteranges media type. The protocol for range requests is an OPTIONAL feature of HTTP, designed so resources or recipients that do not implement this feature can respond as if it is a normal GET request without impacting interoperability. Partial responses are indicated by a distinct status code to not be mistaken for full responses by intermediate caches that might not implement the feature.

Although the HTTP range request mechanism is designed to allow for extensible range types, this specification only defines requests for byte ranges.

1.1. Conformance and Error Handling

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].

This specification targets conformance criteria according to the role of a participant in HTTP communication. Hence, HTTP requirements are placed on senders, recipients, clients, servers, user agents, intermediaries, origin servers, proxies, gateways, or caches, depending on what behavior is being constrained by the requirement. See Section 2 of [Part1] for definitions of these terms.

The verb "generate" is used instead of "send" where a requirement differentiates between creating a protocol element and merely forwarding a received element downstream.

An implementation is considered conformant if it complies with all of the requirements associated with the roles it partakes in HTTP. Note that SHOULD-level requirements are relevant here, unless one of the documented exceptions is applicable.

This document also uses ABNF to define valid protocol elements

(Section 1.2). In addition to the prose requirements placed upon them, senders **MUST NOT** generate protocol elements that do not match the grammar defined by the ABNF rules for those protocol elements that are applicable to the sender's role. If a received protocol element is processed, the recipient **MUST** be able to parse any value that would match the ABNF rules for that protocol element, excluding only those rules not applicable to the recipient's role.

Unless noted otherwise, a recipient **MAY** attempt to recover a usable protocol element from an invalid construct. HTTP does not define specific error handling mechanisms except when they have a direct impact on security, since different applications of the protocol require different error handling strategies. For example, a Web browser might wish to transparently recover from a response where the Location header field doesn't parse according to the ABNF, whereas a systems control client might consider any form of error recovery to be dangerous.

1.2. Syntax Notation

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] with the list rule extension defined in Section 1.2 of [Part1]. Appendix C describes rules imported from other documents. Appendix D shows the collected ABNF with the list rule expanded.

2. Range Units

HTTP/1.1 allows a client to request that only part (a range) of the representation be included within the response. HTTP/1.1 uses range units in the Range (Section 5.4) and Content-Range (Section 5.2) header fields. A representation can be broken down into subranges according to various structural units.

```
range-unit      = bytes-unit / other-range-unit
bytes-unit      = "bytes"
other-range-unit = token
```

HTTP/1.1 has been designed to allow implementations of applications that do not depend on knowledge of ranges. The only range unit defined by HTTP/1.1 is "bytes". Additional specifiers can be defined as described in Section 2.1.

If a range unit is not understood in a request, a server **MUST** ignore the whole Range header field (Section 5.4). If a range unit is not understood in a response, an intermediary **SHOULD** pass the response to the client; a client **MUST** fail.

2.1. Range Specifier Registry

The HTTP Range Specifier Registry defines the name space for the range specifier names.

Registrations MUST include the following fields:

- o Name
- o Description
- o Pointer to specification text

Values to be added to this name space require IETF Review (see [RFC5226], Section 4.1).

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

3. Status Code Definitions

3.1. 206 Partial Content

The server has fulfilled the partial GET request for the resource. The request MUST have included a Range header field (Section 5.4) indicating the desired range, and MAY have included an If-Range header field (Section 5.3) to make the request conditional.

The response MUST include the following header fields:

- o Either a Content-Range header field (Section 5.2) indicating the range included with this response, or a multipart/byteranges Content-Type including Content-Range fields for each part. If a Content-Length header field is present in the response, its value MUST match the actual number of octets transmitted in the message body.
- o Date
- o Cache-Control, ETag, Expires, Content-Location and/or Vary, if the header field would have been sent in a 200 (OK) response to the same request

If a 206 is sent in response to a request with an If-Range header field, it SHOULD NOT include other representation header fields. Otherwise, the response MUST include all of the representation header fields that would have been returned with a 200 (OK) response to the same request.

Caches MAY use a heuristic (see Section 4.1.2 of [Part6]) to determine freshness for 206 responses.

3.2. 416 Requested Range Not Satisfiable

A server SHOULD return a response with this status code if a request included a Range header field (Section 5.4), and none of the ranges-specifier values in this field overlap the current extent of the selected resource, and the request did not include an If-Range header field (Section 5.3). (For byte-ranges, this means that the first-byte-pos of all of the byte-range-spec values were greater than the current length of the selected resource.)

When this status code is returned for a byte-range request, the response SHOULD include a Content-Range header field specifying the current length of the representation (see Section 5.2). This response MUST NOT use the multipart/byteranges content-type. For example,

```
HTTP/1.1 416 Requested Range Not Satisfiable
Date: Mon, 20 Jan 2012 15:41:54 GMT
Content-Range: bytes */47022
Content-Type: image/gif
```

Note: Clients cannot depend on servers to send a 416 (Requested Range Not Satisfiable) response instead of a 200 (OK) response for an unsatisfiable Range header field, since not all servers implement this header field.

4. Responses to a Range Request

4.1. Response to a Single and Multiple Ranges Request

When an HTTP message includes the content of a single range (for example, a response to a request for a single range, or to a request for a set of ranges that overlap without any holes), this content is transmitted with a Content-Range header field, and a Content-Length header field showing the number of bytes actually transferred. For example,

```
HTTP/1.1 206 Partial Content
Date: Wed, 15 Nov 1995 06:25:24 GMT
Last-Modified: Wed, 15 Nov 1995 04:58:08 GMT
Content-Range: bytes 21010-47021/47022
Content-Length: 26012
Content-Type: image/gif
```

When an HTTP message includes the content of multiple ranges (for

example, a response to a request for multiple non-overlapping ranges), these are transmitted as a multipart message. The multipart media type used for this purpose is "multipart/byteranges" as defined in Appendix A.

A server MAY combine requested ranges when those ranges are overlapping (see Section 7.1).

A response to a request for a single range MUST NOT be sent using the multipart/byteranges media type. A response to a request for multiple ranges, whose result is a single range, MAY be sent as a multipart/byteranges media type with one part. A client that cannot decode a multipart/byteranges message MUST NOT ask for multiple ranges in a single request.

When a client asks for multiple ranges in one request, the server SHOULD return them in the order that they appeared in the request.

4.2. Combining Ranges

A response might transfer only a subrange of a representation if the connection closed prematurely or if the request used one or more Range specifications. After several such transfers, a client might have received several ranges of the same representation. These ranges can only be safely combined if they all have in common the same strong validator, where "strong validator" is defined to be either an entity-tag that is not marked as weak (Section 2.3 of [Part4]) or, if no entity-tag is provided, a Last-Modified value that is strong in the sense defined by Section 2.2.2 of [Part4].

When a client receives an incomplete 200 (OK) or 206 (Partial Content) response and already has one or more stored responses for the same method and effective request URI, all of the stored responses with the same strong validator MAY be combined with the partial content in this new response. If none of the stored responses contain the same strong validator, then this new response corresponds to a new representation and MUST NOT be combined with the existing stored responses.

If the new response is an incomplete 200 (OK) response, then the header fields of that new response are used for any combined response and replace those of the matching stored responses.

If the new response is a 206 (Partial Content) response and at least one of the matching stored responses is a 200 (OK), then the combined response header fields consist of the most recent 200 response's header fields. If all of the matching stored responses are 206 responses, then the stored response with the most header fields is

used as the source of header fields for the combined response, except that the client **MUST** use other header fields provided in the new response, aside from Content-Range, to replace all instances of the corresponding header fields in the stored response.

The combined response message body consists of the union of partial content ranges in the new response and each of the selected responses. If the union consists of the entire range of the representation, then the combined response **MUST** be recorded as a complete 200 (OK) response with a Content-Length header field that reflects the complete length. Otherwise, the combined response(s) **MUST** include a Content-Range header field describing the included range(s) and be recorded as incomplete. If the union consists of a discontinuous range of the representation, then the client **MAY** store it as either a multipart range response or as multiple 206 responses with one continuous range each.

5. Header Field Definitions

This section defines the syntax and semantics of HTTP/1.1 header fields related to range requests and partial responses.

5.1. Accept-Ranges

The "Accept-Ranges" header field allows a resource to indicate its acceptance of range requests.

```
Accept-Ranges      = acceptable-ranges
acceptable-ranges = 1#range-unit / "none"
```

Origin servers that accept byte-range requests **MAY** send

```
Accept-Ranges: bytes
```

but are not required to do so. Clients **MAY** generate range requests without having received this header field for the resource involved. Range units are defined in Section 2.

Servers that do not accept any kind of range request for a resource **MAY** send

```
Accept-Ranges: none
```

to advise the client not to attempt a range request.

5.2. Content-Range

The "Content-Range" header field is sent with a partial representation to specify where in the full representation the payload body is intended to be applied.

Range units are defined in Section 2.

```
Content-Range          = byte-content-range-spec
                        / other-content-range-spec

byte-content-range-spec = bytes-unit SP
                        byte-range-resp-spec "/"
                        ( instance-length / "*" )

byte-range-resp-spec    = (first-byte-pos "-" last-byte-pos)
                        / "*"

instance-length          = 1*DIGIT

other-content-range-spec = other-range-unit SP
                        other-range-resp-spec
other-range-resp-spec    = *CHAR
```

The header field SHOULD indicate the total length of the full representation, unless this length is unknown or difficult to determine. The asterisk "*" character means that the instance-length is unknown at the time when the response was generated.

Unlike byte-ranges-specifier values (see Section 5.4.1), a byte-range-resp-spec MUST only specify one range, and MUST contain absolute byte positions for both the first and last byte of the range.

A byte-content-range-spec with a byte-range-resp-spec whose last-byte-pos value is less than its first-byte-pos value, or whose instance-length value is less than or equal to its last-byte-pos value, is invalid. The recipient of an invalid byte-content-range-spec MUST ignore it and any content transferred along with it.

In the case of a byte range request: A server sending a response with status code 416 (Requested Range Not Satisfiable) SHOULD include a Content-Range field with a byte-range-resp-spec of "*". The instance-length specifies the current length of the selected resource. A response with status code 206 (Partial Content) MUST NOT include a Content-Range field with a byte-range-resp-spec of "*".

The "Content-Range" header field has no meaning for status codes that

do not explicitly describe its semantic. Currently, only status codes 206 (Partial Content) and 416 (Requested Range Not Satisfiable) describe the meaning of this header field.

Examples of byte-content-range-spec values, assuming that the representation contains a total of 1234 bytes:

- o The first 500 bytes:

bytes 0-499/1234

- o The second 500 bytes:

bytes 500-999/1234

- o All except for the first 500 bytes:

bytes 500-1233/1234

- o The last 500 bytes:

bytes 734-1233/1234

If the server ignores a byte-range-spec (for example if it is syntactically invalid, or if it might be seen as a denial-of-service attack), the server SHOULD treat the request as if the invalid Range header field did not exist. (Normally, this means return a 200 (OK) response containing the full representation).

5.3. If-Range

If a client has a partial copy of a representation and wishes to have an up-to-date copy of the entire representation, it could use the Range header field with a conditional GET (using either or both of If-Unmodified-Since and If-Match.) However, if the condition fails because the representation has been modified, the client would then have to make a second request to obtain the entire current representation.

The "If-Range" header field allows a client to "short-circuit" the second request. Informally, its meaning is "if the representation is unchanged, send me the part(s) that I am missing; otherwise, send me the entire new representation".

If-Range = entity-tag / HTTP-date

Clients MUST NOT use an entity-tag marked as weak in an If-Range field value and MUST NOT use a Last-Modified date in an If-Range

field value unless it has no entity-tag for the representation and the Last-Modified date it does have for the representation is strong in the sense defined by Section 2.2.2 of [Part4].

A server that evaluates a conditional range request that is applicable to one of its representations MUST evaluate the condition as false if the entity-tag used as a validator is marked as weak or, when an HTTP-date is used as the validator, if the date value is not strong in the sense defined by Section 2.2.2 of [Part4]. (A server can distinguish between a valid HTTP-date and any form of entity-tag by examining the first two characters.)

The If-Range header field SHOULD only be sent by clients together with a Range header field. The If-Range header field MUST be ignored if it is received in a request that does not include a Range header field. The If-Range header field MUST be ignored by a server that does not support the sub-range operation.

If the validator given in the If-Range header field matches the current validator for the selected representation of the target resource, then the server SHOULD send the specified sub-range of the representation using a 206 (Partial Content) response. If the validator does not match, then the server SHOULD send the entire representation using a 200 (OK) response.

5.4. Range

5.4.1. Byte Ranges

Since all HTTP representations are transferred as sequences of bytes, the concept of a byte range is meaningful for any HTTP representation. (However, not all clients and servers need to support byte-range operations.)

Byte range specifications in HTTP apply to the sequence of bytes in the representation body (not necessarily the same as the message body).

A byte range operation MAY specify a single range of bytes, or a set of ranges within a single representation.

```
byte-ranges-specifier = bytes-unit "=" byte-range-set
byte-range-set       = 1#( byte-range-spec / suffix-byte-range-spec )
byte-range-spec      = first-byte-pos "-" [ last-byte-pos ]
first-byte-pos       = 1*DIGIT
last-byte-pos        = 1*DIGIT
```

The first-byte-pos value in a byte-range-spec gives the byte-offset

of the first byte in a range. The last-byte-pos value gives the byte-offset of the last byte in the range; that is, the byte positions specified are inclusive. Byte offsets start at zero.

If the last-byte-pos value is present, it MUST be greater than or equal to the first-byte-pos in that byte-range-spec, or the byte-range-spec is syntactically invalid. The recipient of a byte-range-set that includes one or more syntactically invalid byte-range-spec values MUST ignore the header field that includes that byte-range-set.

If the last-byte-pos value is absent, or if the value is greater than or equal to the current length of the representation body, last-byte-pos is taken to be equal to one less than the current length of the representation in bytes.

By its choice of last-byte-pos, a client can limit the number of bytes retrieved without knowing the size of the representation.

```
suffix-byte-range-spec = "-" suffix-length  
suffix-length = 1*DIGIT
```

A suffix-byte-range-spec is used to specify the suffix of the representation body, of a length given by the suffix-length value. (That is, this form specifies the last N bytes of a representation.) If the representation is shorter than the specified suffix-length, the entire representation is used.

If a syntactically valid byte-range-set includes at least one byte-range-spec whose first-byte-pos is less than the current length of the representation, or at least one suffix-byte-range-spec with a non-zero suffix-length, then the byte-range-set is satisfiable. Otherwise, the byte-range-set is unsatisfiable. If the byte-range-set is unsatisfiable, the server SHOULD return a response with a 416 (Requested Range Not Satisfiable) status code. Otherwise, the server SHOULD return a response with a 206 (Partial Content) status code containing the satisfiable ranges of the representation.

In the byte range syntax, first-byte-pos, last-byte-pos, and suffix-length are expressed as decimal number of octets. Since there is no predefined limit to the length of an HTTP payload, recipients SHOULD anticipate potentially large decimal numerals and prevent parsing errors due to integer conversion overflows.

Examples of byte-ranges-specifier values (assuming a representation of length 10000):

- o The first 500 bytes (byte offsets 0-499, inclusive):
bytes=0-499
- o The second 500 bytes (byte offsets 500-999, inclusive):
bytes=500-999
- o The final 500 bytes (byte offsets 9500-9999, inclusive):
bytes=-500

Or:

- bytes=9500-
- o The first and last bytes only (bytes 0 and 9999):
bytes=0-0,-1
- o Several legal but not canonical specifications of the second 500 bytes (byte offsets 500-999, inclusive):
bytes=500-600,601-999
bytes=500-700,601-999

5.4.2. Range Retrieval Requests

The "Range" header field defines the GET method (conditional or not) to request one or more sub-ranges of the response representation body, instead of the entire representation body.

```
Range = byte-ranges-specifier / other-ranges-specifier
other-ranges-specifier = other-range-unit "=" other-range-set
other-range-set = 1*CHAR
```

A server MAY ignore the Range header field. However, origin servers and intermediate caches ought to support byte ranges when possible, since Range supports efficient recovery from partially failed transfers, and supports efficient partial retrieval of large representations.

If the server supports the Range header field and the specified range or ranges are appropriate for the representation:

- o The presence of a Range header field in an unconditional GET modifies what is returned if the GET is otherwise successful. In other words, the response carries a status code of 206 (Partial

Content) instead of 200 (OK).

- o The presence of a Range header field in a conditional GET (a request using one or both of If-Modified-Since and If-None-Match, or one or both of If-Unmodified-Since and If-Match) modifies what is returned if the GET is otherwise successful and the condition is true. It does not affect the 304 (Not Modified) response returned if the conditional is false.

In some cases, it might be more appropriate to use the If-Range header field (see Section 5.3) in addition to the Range header field.

If a proxy that supports ranges receives a Range request, forwards the request to an inbound server, and receives an entire representation in reply, it MAY only return the requested range to its client.

6. IANA Considerations

6.1. Status Code Registration

The HTTP Status Code Registry located at <http://www.iana.org/assignments/http-status-codes> shall be updated with the registrations below:

Value	Description	Reference
206	Partial Content	Section 3.1
416	Requested Range Not Satisfiable	Section 3.2

6.2. Header Field Registration

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

Header Field Name	Protocol	Status	Reference
Accept-Ranges	http	standard	Section 5.1
Content-Range	http	standard	Section 5.2
If-Range	http	standard	Section 5.3
Range	http	standard	Section 5.4

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

Engineering Task Force".

6.3. Range Specifier Registration

The registration procedure for HTTP Range Specifiers is defined by Section 2.1 of this document.

The HTTP Range Specifier Registry shall be created at <http://www.iana.org/assignments/http-range-specifiers> and be populated with the registrations below:

Range Specifier Name	Description	Reference
bytes	a range of octets	Section 2
none	reserved as keyword, indicating no ranges are supported	Section 5.1

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

7. Security Considerations

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

7.1. Overlapping Ranges

Range requests containing overlapping ranges can lead to the situation where a server is sending far more data than the size of the complete resource representation.

8. Acknowledgments

See Section 9 of [Part1].

9. References

9.1. Normative References

[Part1] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: Message Routing and Syntax",

draft-ietf-httpbis-p1-messaging-20 (work in progress),
July 2012.

[Part2] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed.,
"HTTP/1.1, part 2: Semantics and Payloads",
draft-ietf-httpbis-p2-semantics-20 (work in progress),
July 2012.

[Part4] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed.,
"HTTP/1.1, part 4: Conditional Requests",
draft-ietf-httpbis-p4-conditional-20 (work in progress),
July 2012.

[Part6] Fielding, R., Ed., Lafon, Y., Ed., Nottingham, M., Ed.,
and J. Reschke, Ed., "HTTP/1.1, part 6: Caching",
draft-ietf-httpbis-p6-cache-20 (work in progress),
July 2012.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail
Extensions (MIME) Part Two: Media Types", RFC 2046,
November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax
Specifications: ABNF", STD 68, RFC 5234, January 2008.

9.2. Informative References

[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.

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

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

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

Appendix A. Internet Media Type multipart/byteranges

When an HTTP 206 (Partial Content) response message includes the content of multiple ranges (a response to a request for multiple non-overlapping ranges), these are transmitted as a multipart message body ([RFC2046], Section 5.1). The media type for this purpose is called "multipart/byteranges". The following is to be registered with IANA [RFC4288].

The multipart/byteranges media type includes one or more parts, each with its own Content-Type and Content-Range fields. The required boundary parameter specifies the boundary string used to separate each body-part.

Type name: multipart

Subtype name: byteranges

Required parameters: boundary

Optional parameters: none

Encoding considerations: only "7bit", "8bit", or "binary" are permitted

Security considerations: none

Interoperability considerations: none

Published specification: This specification (see Appendix A).

Applications that use this media type: HTTP components supporting multiple ranges in a single request.

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

Note: Despite the name "multipart/byteranges" is not limited to the byte ranges only.

For example:

```
HTTP/1.1 206 Partial Content
Date: Wed, 15 Nov 1995 06:25:24 GMT
Last-Modified: Wed, 15 Nov 1995 04:58:08 GMT
Content-type: multipart/byteranges; boundary=THIS_STRING_SEPARATES
```

```
--THIS_STRING_SEPARATES
Content-type: application/pdf
Content-range: bytes 500-999/8000
```

```
...the first range...
--THIS_STRING_SEPARATES
Content-type: application/pdf
Content-range: bytes 7000-7999/8000
```

```
...the second range
--THIS_STRING_SEPARATES--
```

Another example, using the "exampleunit" range unit:

```
HTTP/1.1 206 Partial Content
Date: Tue, 14 Nov 1995 06:25:24 GMT
Last-Modified: Tue, 14 July 04:58:08 GMT
Content-type: multipart/byteranges; boundary=THIS_STRING_SEPARATES
```

```
--THIS_STRING_SEPARATES
Content-type: video/example
Content-range: exampleunit 1.2-4.3/25
```

```
...the first range...
--THIS_STRING_SEPARATES
Content-type: video/example
Content-range: exampleunit 11.2-14.3/25
```

```
...the second range
--THIS_STRING_SEPARATES--
```

Notes:

1. Additional CRLFs MAY precede the first boundary string in the body.
2. Although [RFC2046] permits the boundary string to be quoted, some existing implementations handle a quoted boundary string incorrectly.
3. A number of clients and servers were coded to an early draft of the byteranges specification to use a media type of multipart/x-byteranges, which is almost, but not quite compatible with the version documented in HTTP/1.1.

Appendix B. Changes from RFC 2616

Introduce Range Specifier Registry. (Section 2.1)

Clarify that it is not ok to use a weak validator in a 206 response. (Section 3.1)

Change ABNF productions for header fields to only define the field value. (Section 5)

Clarify that multipart/byteranges can consist of a single part. (Appendix A)

Appendix C. Imported ABNF

The following core rules are included by reference, as defined in Appendix B.1 of [RFC5234]: 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), and VCHAR (any visible US-ASCII character).

Note that all rules derived from token are to be compared case-insensitively, like range-unit and acceptable-ranges.

The rules below are defined in [Part1]:

OWS	= <OWS, defined in [Part1], Section 3.2.1>
token	= <token, defined in [Part1], Section 3.2.4>

The rules below are defined in other parts:

HTTP-date	= <HTTP-date, defined in [Part2], Section 5.1>
entity-tag	= <entity-tag, defined in [Part4], Section 2.3>

Appendix D. Collected ABNF

```
Accept-Ranges = acceptable-ranges

Content-Range = byte-content-range-spec / other-content-range-spec

HTTP-date = <HTTP-date, defined in [Part2], Section 5.1>

If-Range = entity-tag / HTTP-date

OWS = <OWS, defined in [Part1], Section 3.2.1>

Range = byte-ranges-specifier / other-ranges-specifier

acceptable-ranges = ( *( "," OWS ) range-unit *( OWS "," [ OWS
    range-unit ] ) ) / "none"

byte-content-range-spec = bytes-unit SP byte-range-resp-spec "/" (
    instance-length / "*" )
byte-range-resp-spec = ( first-byte-pos "-" last-byte-pos ) / "*"
byte-range-set = *( "," OWS ) ( byte-range-spec /
    suffix-byte-range-spec ) *( OWS "," [ OWS ( byte-range-spec /
    suffix-byte-range-spec ) ] )
byte-range-spec = first-byte-pos "-" [ last-byte-pos ]
byte-ranges-specifier = bytes-unit "=" byte-range-set
bytes-unit = "bytes"

entity-tag = <entity-tag, defined in [Part4], Section 2.3>

first-byte-pos = 1*DIGIT

instance-length = 1*DIGIT

last-byte-pos = 1*DIGIT

other-content-range-spec = other-range-unit SP other-range-resp-spec
other-range-resp-spec = *CHAR
other-range-set = 1*CHAR
other-range-unit = token
other-ranges-specifier = other-range-unit "=" other-range-set

range-unit = bytes-unit / other-range-unit

suffix-byte-range-spec = "-" suffix-length
suffix-length = 1*DIGIT

token = <token, defined in [Part1], Section 3.2.4>
```

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

Changes up to the first Working Group Last Call draft are summarized in <<http://tools.ietf.org/html/draft-ietf-httpbis-p5-range-19#appendix-D>>.

E.1. Since draft-ietf-httpbis-p5-range-19

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/358>>: "ABNF list expansion code problem"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/361>>: "ABNF requirements for recipients"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/367>>: "reserve 'none' as byte range unit"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/368>>: "note introduction of new IANA registries as normative changes"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/369>>: "range units vs leading zeroes vs size"

Index

2	206 Partial Content (status code)	6
4	416 Requested Range Not Satisfiable (status code)	7
A	Accept-Ranges header field	9
C	Content-Range header field	10
G	Grammar	
	Accept-Ranges	9
	acceptable-ranges	9
	byte-content-range-spec	10
	byte-range-resp-spec	10
	byte-range-set	12
	byte-range-spec	12
	byte-ranges-specifier	12

- bytes-unit 5
- Content-Range 10
- first-byte-pos 12
- If-Range 11
- instance-length 10
- last-byte-pos 12
- other-range-unit 5
- Range 14
- range-unit 5
- ranges-specifier 12
- suffix-byte-range-spec 13
- suffix-length 13

H

Header Fields

- Accept-Ranges 9
- Content-Range 10
- If-Range 11
- Range 12

I

- If-Range header field 11

M

Media Type

- multipart/byteranges 18
- multipart/x-byteranges 20
- multipart/byteranges Media Type 18
- multipart/x-byteranges Media Type 20

R

- Range header field 12

S

Status Codes

- 206 Partial Content 6
- 416 Requested Range Not Satisfiable 7

Authors' Addresses

Roy T. Fielding (editor)
Adobe Systems Incorporated
345 Park Ave
San Jose, CA 95110
USA

EMail: fielding@gbiv.com
URI: <http://roy.gbiv.com/>

Yves Lafon (editor)
World Wide Web Consortium
W3C / ERCIM
2004, rte des Lucioles
Sophia-Antipolis, AM 06902
France

EMail: ylafon@w3.org
URI: <http://www.raubacapeu.net/people/yves/>

Julian F. Reschke (editor)
greenbytes GmbH
Hafenweg 16
Muenster, NW 48155
Germany

EMail: julian.reschke@greenbytes.de
URI: <http://greenbytes.de/tech/webdav/>

HTTPbis Working Group
Internet-Draft
Obsoletes: 2616 (if approved)
Intended status: Standards Track
Expires: January 17, 2013

R. Fielding, Ed.
Adobe
Y. Lafon, Ed.
W3C
M. Nottingham, Ed.
Rackspace
J. Reschke, Ed.
greenbytes
July 16, 2012

HTTP/1.1, part 6: Caching
draft-ietf-httpbis-p6-cache-20

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypertext information systems. This document defines requirements on HTTP caches and the associated header fields that control cache behavior or indicate cacheable response messages.

Editorial Note (To be removed by RFC Editor)

Discussion of this draft takes place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org), which is archived at <http://lists.w3.org/Archives/Public/ietf-http-wg/>.

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 D.1.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference

material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 17, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	4
1.1. Purpose	4
1.2. Terminology	4
1.3. Conformance and Error Handling	6
1.4. Syntax Notation	7
1.4.1. Delta Seconds	7
2. Overview of Cache Operation	7
3. Storing Responses in Caches	8
3.1. Storing Incomplete Responses	9
3.2. Storing Responses to Authenticated Requests	9
4. Constructing Responses from Caches	10
4.1. Freshness Model	11
4.1.1. Calculating Freshness Lifetime	12
4.1.2. Calculating Heuristic Freshness	12
4.1.3. Calculating Age	13

4.1.4. Serving Stale Responses	15
4.2. Validation Model	16
4.2.1. Freshening Responses with 304 Not Modified	16
4.3. Using Negotiated Responses	17
4.4. Combining Partial Content	18
5. Updating Caches with HEAD Responses	19
6. Request Methods that Invalidate	19
7. Header Field Definitions	20
7.1. Age	20
7.2. Cache-Control	20
7.2.1. Request Cache-Control Directives	21
7.2.2. Response Cache-Control Directives	23
7.2.3. Cache Control Extensions	26
7.3. Expires	28
7.4. Pragma	28
7.5. Vary	29
7.6. Warning	30
7.6.1. 110 Response is Stale	31
7.6.2. 111 Revalidation Failed	32
7.6.3. 112 Disconnected Operation	32
7.6.4. 113 Heuristic Expiration	32
7.6.5. 199 Miscellaneous Warning	32
7.6.6. 214 Transformation Applied	32
7.6.7. 299 Miscellaneous Persistent Warning	32
7.6.8. Warn Code Extensions	32
8. History Lists	33
9. IANA Considerations	33
9.1. Cache Directive Registry	33
9.2. Warn Code Registry	34
9.3. Header Field Registration	34
10. Security Considerations	35
11. Acknowledgments	35
12. References	35
12.1. Normative References	35
12.2. Informative References	36
Appendix A. Changes from RFC 2616	36
Appendix B. Imported ABNF	37
Appendix C. Collected ABNF	38
Appendix D. Change Log (to be removed by RFC Editor before publication)	39
D.1. Since draft-ietf-httpbis-p6-cache-19	39
Index	39

1. Introduction

HTTP is typically used for distributed information systems, where performance can be improved by the use of response caches. This document defines aspects of HTTP/1.1 related to caching and reusing response messages.

1.1. Purpose

An HTTP cache is a local store of 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 that is acting as a tunnel.

The goal of caching in HTTP/1.1 is to significantly improve performance by reusing a prior response message to satisfy a current request. A stored response is considered "fresh", as defined in Section 4.1, if the response can be reused without "validation" (checking with the origin server to see if the cached response remains valid for this request). A fresh cache response can therefore reduce both latency and network transfers each time it is reused. When a cached response is not fresh, it might still be reusable if it can be freshened by validation (Section 4.2) or if the origin is unavailable.

1.2. Terminology

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

cache

A conformant implementation of a HTTP cache. Note that this implies an HTTP/1.1 cache; this specification does not define conformance for HTTP/1.0 caches.

shared cache

A cache that stores responses to be reused by more than one user; usually (but not always) deployed as part of an intermediary.

private cache

A cache that is dedicated to a single user.

cacheable

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 on whether a cache can use the stored copy to satisfy a particular request.

explicit expiration time

The time at which the origin server intends that a representation no longer be returned by a cache without further validation.

heuristic expiration time

An expiration time assigned by a cache when no explicit expiration time is available.

age

The age of a response is the time since it was sent by, or successfully validated with, the origin server.

first-hand

A response is first-hand if the freshness model is not in use; i.e., its age is 0.

freshness lifetime

The length of time between the generation of a response and its expiration time.

fresh

A response is fresh if its age has not yet exceeded its freshness lifetime.

stale

A response is stale if its age has passed its freshness lifetime (either explicit or heuristic).

validator

A protocol element (e.g., an entity-tag or a Last-Modified time) that is used to find out whether a stored response is an equivalent copy of a representation. See Section 2.1 of [Part4].

strong validator

A validator that is defined by the origin server such that its current value will change if the representation body changes; i.e., an entity-tag that is not marked as weak (Section 2.3 of [Part4]) or, if no entity-tag is provided, a Last-Modified value that is strong in the sense defined by Section 2.2.2 of [Part4].

1.3. Conformance and Error Handling

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].

This specification targets conformance criteria according to the role of a participant in HTTP communication. Hence, HTTP requirements are placed on senders, recipients, clients, servers, user agents, intermediaries, origin servers, proxies, gateways, or caches, depending on what behavior is being constrained by the requirement. See Section 2 of [Part1] for definitions of these terms.

The verb "generate" is used instead of "send" where a requirement differentiates between creating a protocol element and merely forwarding a received element downstream.

An implementation is considered conformant if it complies with all of the requirements associated with the roles it partakes in HTTP. Note that SHOULD-level requirements are relevant here, unless one of the documented exceptions is applicable.

This document also uses ABNF to define valid protocol elements (Section 1.4). In addition to the prose requirements placed upon them, senders MUST NOT generate protocol elements that do not match the grammar defined by the ABNF rules for those protocol elements that are applicable to the sender's role. If a received protocol element is processed, the recipient MUST be able to parse any value that would match the ABNF rules for that protocol element, excluding only those rules not applicable to the recipient's role.

Unless noted otherwise, a recipient MAY attempt to recover a usable protocol element from an invalid construct. HTTP does not define specific error handling mechanisms except when they have a direct impact on security, since different applications of the protocol require different error handling strategies. For example, a Web browser might wish to transparently recover from a response where the Location header field doesn't parse according to the ABNF, whereas a systems control client might consider any form of error recovery to be dangerous.

1.4. Syntax Notation

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] with the list rule extension defined in Section 1.2 of [Part1]. Appendix B describes rules imported from other documents. Appendix C shows the collected ABNF with the list rule expanded.

1.4.1. Delta Seconds

The delta-seconds rule specifies a non-negative integer, representing time in seconds.

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

If an implementation receives a delta-seconds value larger than the largest positive integer it can represent, or if any of its subsequent calculations overflows, it MUST consider the value to be 2147483648 (2^{31}). Recipients parsing a delta-seconds value MUST use an arithmetic type of at least 31 bits of range, and senders MUST NOT send delta-seconds with a value greater than 2147483648.

2. Overview of Cache Operation

Proper cache operation preserves the semantics of HTTP transfers ([Part2]) while eliminating the transfer of information already held in the cache. Although caching is an entirely OPTIONAL feature of HTTP, we assume that reusing the cached response is desirable and that such reuse is the default behavior when no requirement or locally-desired configuration prevents it. Therefore, HTTP cache requirements are focused on preventing a cache from either storing a non-reusable response or reusing a stored response inappropriately.

Each cache entry consists of a cache key and one or more HTTP responses corresponding to prior requests that used the same key. The most common form of cache entry is a successful result of a retrieval request: i.e., a 200 (OK) response containing a representation of the resource identified by the request target. However, it is also possible to cache negative results (e.g., 404 (Not Found), incomplete results (e.g., 206 (Partial Content)), and responses to methods other than GET if the method's definition allows such caching and defines something suitable for use as a cache key.

The default cache key consists of the request method and target URI. However, since HTTP caches in common use today are typically limited to caching responses to GET, many implementations simply decline other methods and use only the URI as the key.

If a request target is subject to content negotiation, its cache entry might consist of multiple stored responses, each differentiated by a secondary key for the values of the original request's selecting header fields (Section 4.3).

3. Storing Responses in Caches

A cache **MUST NOT** store a response to any request, unless:

- o The request method is understood by the cache and defined as being cacheable, and
- o the response status code is understood by the cache, and
- o the "no-store" cache directive (see Section 7.2) does not appear in request or response header fields, and
- o the "private" cache response directive (see Section 7.2.2.2) does not appear in the response, if the cache is shared, and
- o the Authorization header field (see Section 4.1 of [Part7]) does not appear in the request, if the cache is shared, unless the response explicitly allows it (see Section 3.2), and
- o the response either:
 - * contains an Expires header field (see Section 7.3), or
 - * contains a max-age response cache directive (see Section 7.2.2.7), or
 - * contains a s-maxage response cache directive and the cache is shared, or
 - * contains a Cache Control Extension (see Section 7.2.3) that allows it to be cached, or
 - * has a status code that can be served with heuristic freshness (see Section 4.1.2).

Note that any of the requirements listed above can be overridden by a cache-control extension; see Section 7.2.3.

In this context, a cache has "understood" a request method or a response status code if it recognizes it and implements any cache-specific behavior.

Note that, in normal operation, many caches will not store a response

that has neither a cache validator nor an explicit expiration time, as such responses are not usually useful to store. However, caches are not prohibited from storing such responses.

3.1. Storing Incomplete Responses

A response message is considered complete when all of the octets indicated by the message framing ([Part1]) are received prior to the connection being closed. If the request is GET, the response status is 200 (OK), and the entire response header block has been received, a cache MAY store an incomplete response message body if the cache entry is recorded as incomplete. Likewise, a 206 (Partial Content) response MAY be stored as if it were an incomplete 200 (OK) cache entry. However, a cache MUST NOT store incomplete or partial content responses if it does not support the Range and Content-Range header fields or if it does not understand the range units used in those fields.

A cache MAY complete a stored incomplete response by making a subsequent range request ([Part5]) and combining the successful response with the stored entry, as defined in Section 4.4. A cache MUST NOT use an incomplete response to answer requests unless the response has been made complete or the request is partial and specifies a range that is wholly within the incomplete response. A cache MUST NOT send a partial response to a client without explicitly marking it as such using the 206 (Partial Content) status code.

3.2. Storing Responses to Authenticated Requests

A shared cache MUST NOT use a cached response to a request with an Authorization header field (Section 4.1 of [Part7]) to satisfy any subsequent request unless a cache directive that allows such responses to be stored is present in the response.

In this specification, the following Cache-Control response directives (Section 7.2.2) have such an effect: must-revalidate, public, s-maxage.

Note that cached responses that contain the "must-revalidate" and/or "s-maxage" response directives are not allowed to be served stale (Section 4.1.4) by shared caches. In particular, a response with either "max-age=0, must-revalidate" or "s-maxage=0" cannot be used to satisfy a subsequent request without revalidating it on the origin server.

4. Constructing Responses from Caches

For a presented request, a cache MUST NOT return a stored response, unless:

- o The presented effective request URI (Section 5.5 of [Part1]) and that of the stored response match, and
- o the request method associated with the stored response allows it to be used for the presented request, and
- o selecting header fields nominated by the stored response (if any) match those presented (see Section 4.3), and
- o the presented request does not contain the no-cache pragma (Section 7.4), nor the no-cache cache directive (Section 7.2.1), unless the stored response is successfully validated (Section 4.2), and
- o the stored response does not contain the no-cache cache directive (Section 7.2.2.3), unless it is successfully validated (Section 4.2), and
- o the stored response is either:
 - * fresh (see Section 4.1), or
 - * allowed to be served stale (see Section 4.1.4), or
 - * successfully validated (see Section 4.2).

Note that any of the requirements listed above can be overridden by a cache-control extension; see Section 7.2.3.

When a stored response is used to satisfy a request without validation, a cache MUST include a single Age header field (Section 7.1) in the response with a value equal to the stored response's current_age; see Section 4.1.3.

A cache MUST write through requests with methods that are unsafe (Section 2.1.1 of [Part2]) to the origin server; i.e., a cache is not allowed to generate a reply to such a request before having forwarded the request and having received a corresponding response.

Also, note that unsafe requests might invalidate already stored responses; see Section 6.

When more than one suitable response is stored, a cache MUST use the

most recent response (as determined by the Date header field). It can also forward a request with "Cache-Control: max-age=0" or "Cache-Control: no-cache" to disambiguate which response to use.

A cache that does not have a clock available MUST NOT use stored responses without revalidating them on every use. A cache, especially a shared cache, SHOULD use a mechanism, such as NTP [RFC1305], to synchronize its clock with a reliable external standard.

4.1. Freshness Model

When a response is "fresh" in the cache, it can be used to satisfy subsequent requests without contacting the origin server, thereby improving efficiency.

The primary mechanism for determining freshness is for an origin server to provide an explicit expiration time in the future, using either the Expires header field (Section 7.3) or the max-age response cache directive (Section 7.2.2.7). Generally, origin servers will assign future explicit expiration times to responses in the belief that the representation is not likely to change in a semantically significant way before the expiration time is reached.

If an origin server wishes to force a cache to validate every request, it can assign an explicit expiration time in the past to indicate that the response is already stale. Compliant caches will normally validate the cached response before reusing it for subsequent requests (see Section 4.1.4).

Since origin servers do not always provide explicit expiration times, a cache MAY assign a heuristic expiration time when an explicit time is not specified, employing algorithms that use other header field values (such as the Last-Modified time) to estimate a plausible expiration time. This specification does not provide specific algorithms, but does impose worst-case constraints on their results.

The calculation to determine if a response is fresh is:

```
response_is_fresh = (freshness_lifetime > current_age)
```

The `freshness_lifetime` is defined in Section 4.1.1; the `current_age` is defined in Section 4.1.3.

Additionally, clients can influence freshness calculation -- either constraining it relaxing it -- by using the max-age and min-fresh request cache directives. See Section 7.2.1 for details.

Note that freshness applies only to cache operation; it cannot be used to force a user agent to refresh its display or reload a resource. See Section 8 for an explanation of the difference between caches and history mechanisms.

4.1.1. Calculating Freshness Lifetime

A cache can calculate the freshness lifetime (denoted as `freshness_lifetime`) of a response by using the first match of:

- o If the cache is shared and the `s-maxage` response cache directive (Section 7.2.2.8) is present, use its value, or
- o If the `max-age` response cache directive (Section 7.2.2.7) is present, use its value, or
- o If the `Expires` response header field (Section 7.3) is present, use its value minus the value of the `Date` response header field, or
- o Otherwise, no explicit expiration time is present in the response. A heuristic freshness lifetime might be applicable; see Section 4.1.2.

Note that this calculation is not vulnerable to clock skew, since all of the information comes from the origin server.

When there is more than one value present for a given directive (e.g., two `Expires` header fields, multiple `Cache-Control: max-age` directives), it is considered invalid. Caches are encouraged to consider responses that have invalid freshness information to be stale.

4.1.2. Calculating Heuristic Freshness

If no explicit expiration time is present in a stored response that has a status code whose definition allows heuristic freshness to be used (including the following in Section 4 of [Part2]: 200 (OK), 203 (Non-Authoritative Information), 206 (Partial Content), 300 (Multiple Choices), 301 (Moved Permanently) and 410 (Gone)), a cache MAY calculate a heuristic expiration time. A cache MUST NOT use heuristics to determine freshness for responses with status codes that do not explicitly allow it.

When a heuristic is used to calculate freshness lifetime, a cache SHOULD attach a `Warning` header field with a 113 warn-code to the response if its `current_age` is more than 24 hours and such a warning is not already present.

Also, if the response has a Last-Modified header field (Section 2.2 of [Part4]), caches are encouraged to use a heuristic expiration value that is no more than some fraction of the interval since that time. A typical setting of this fraction might be 10%.

Note: Section 13.9 of [RFC2616] prohibited caches from calculating heuristic freshness for URIs with query components (i.e., those containing '?'). In practice, this has not been widely implemented. Therefore, servers are encouraged to send explicit directives (e.g., Cache-Control: no-cache) if they wish to preclude caching.

4.1.3. Calculating Age

HTTP/1.1 uses the Age header field to convey the estimated age of the response message when obtained from a cache. The Age field value is the cache's estimate of the amount of time since the response was generated or validated by the origin server. In essence, the Age value is the sum of the time that the response has been resident in each of the caches along the path from the origin server, plus the amount of time it has been in transit along network paths.

The following data is used for the age calculation:

age_value

The term "age_value" denotes the value of the Age header field (Section 7.1), in a form appropriate for arithmetic operation; or 0, if not available.

date_value

HTTP/1.1 requires origin servers to send a Date header field, if possible, with every response, giving the time at which the response was generated. The term "date_value" denotes the value of the Date header field, in a form appropriate for arithmetic operations. See Section 9.10 of [Part2] for the definition of the Date header field, and for requirements regarding responses without it.

now

The term "now" means "the current value of the clock at the host performing the calculation". A cache SHOULD use NTP ([RFC1305]) or some similar protocol to synchronize its clocks to a globally accurate time standard.

request_time

The current value of the clock at the host at the time the request resulting in the stored response was made.

response_time

The current value of the clock at the host at the time the response was received.

A response's age can be calculated in two entirely independent ways:

1. the "apparent_age": response_time minus date_value, if the local clock is reasonably well synchronized to the origin server's clock. If the result is negative, the result is replaced by zero.
2. the "corrected_age_value", if all of the caches along the response path implement HTTP/1.1. A cache MUST interpret this value relative to the time the request was initiated, not the time that the response was received.

```
apparent_age = max(0, response_time - date_value);
```

```
response_delay = response_time - request_time;  
corrected_age_value = age_value + response_delay;
```

These SHOULD be combined as

```
corrected_initial_age = max(apparent_age, corrected_age_value);
```

unless the cache is confident in the value of the Age header field (e.g., because there are no HTTP/1.0 hops in the Via header field), in which case the corrected_age_value MAY be used as the corrected_initial_age.

The current_age of a stored response can then be calculated by adding the amount of time (in seconds) since the stored response was last validated by the origin server to the corrected_initial_age.

```
resident_time = now - response_time;  
current_age = corrected_initial_age + resident_time;
```

Additionally, to avoid common problems in date parsing:

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

4.1.4. Serving Stale Responses

A "stale" response is one that either has explicit expiry information or is allowed to have heuristic expiry calculated, but is not fresh according to the calculations in Section 4.1.

A cache MUST NOT return a stale response if it is prohibited by an explicit in-protocol directive (e.g., by a "no-store" or "no-cache" cache directive, a "must-revalidate" cache-response-directive, or an applicable "s-maxage" or "proxy-revalidate" cache-response-directive; see Section 7.2.2).

A cache MUST NOT return stale responses unless it is disconnected (i.e., it cannot contact the origin server or otherwise find a forward path) or doing so is explicitly allowed (e.g., by the max-stale request directive; see Section 7.2.1).

A cache SHOULD append a Warning header field with the 110 warn-code (see Section 7.6) to stale responses. Likewise, a cache SHOULD add the 112 warn-code to stale responses if the cache is disconnected.

If a cache receives a first-hand response (either an entire response, or a 304 (Not Modified) response) that it would normally forward to the requesting client, and the received response is no longer fresh, the cache can forward it to the requesting client without adding a new Warning (but without removing any existing Warning header fields). A cache shouldn't attempt to validate a response simply

because that response became stale in transit.

4.2. Validation Model

When a cache has one or more stored responses for a requested URI, but cannot serve any of them (e.g., because they are not fresh, or one cannot be selected; see Section 4.3), it can use the conditional request mechanism [Part4] in the forwarded request to give the origin server an opportunity to both select a valid stored response to be used, and to update it. This process is known as "validating" or "revalidating" the stored response.

When sending such a conditional request, a cache adds an If-Modified-Since header field whose value is that of the Last-Modified header field from the selected (see Section 4.3) stored response, if available.

Additionally, a cache can add an If-None-Match header field whose value is that of the ETag header field(s) from all responses stored for the requested URI, if present. However, if any of the stored responses contains only partial content, the cache shouldn't include its entity-tag in the If-None-Match header field unless the request is for a range that would be fully satisfied by that stored response.

Cache handling of a response to a conditional request is dependent upon its status code:

- o A 304 (Not Modified) response status code indicates that the stored response can be updated and reused; see Section 4.2.1.
- o A full response (i.e., one with a response body) indicates that none of the stored responses nominated in the conditional request is suitable. Instead, the cache can use the full response to satisfy the request and MAY replace the stored response(s).
- o However, if a cache receives a 5xx (Server Error) response while attempting to validate a response, it can either forward this response to the requesting client, or act as if the server failed to respond. In the latter case, it can return a previously stored response (see Section 4.1.4).

4.2.1. Freshening Responses with 304 Not Modified

When a cache receives a 304 (Not Modified) response and already has one or more stored 200 (OK) responses for the same cache key, the cache needs to identify which of the stored responses are updated by this new response and then update the stored response(s) with the new information provided in the 304 response.

- o If the new response contains a strong validator, then that strong validator identifies the selected representation. All of the stored responses with the same strong validator are selected. If none of the stored responses contain the same strong validator, then this new response corresponds to a new selected representation and MUST NOT update the existing stored responses.
- o If the new response contains a weak validator and that validator corresponds to one of the cache's stored responses, then the most recent of those matching stored responses is selected.
- o If the new response does not include any form of validator, there is only one stored response, and that stored response also lacks a validator, then that stored response is selected.

If a stored response is selected for update, the cache MUST:

- o delete any Warning header fields in the stored response with warn-code 1xx (see Section 7.6);
- o retain any Warning header fields in the stored response with warn-code 2xx; and,
- o use other header fields provided in the 304 (Not Modified) response to replace all instances of the corresponding header fields in the stored response.

4.3. Using Negotiated Responses

When a cache receives a request that can be satisfied by a stored response that has a Vary header field (Section 7.5), it MUST NOT use that response unless all of the selecting header fields nominated by the Vary header field match in both the original request (i.e., that associated with the stored response), and the presented request.

The selecting header fields from two requests are defined to match if and only if those in the first request can be transformed to those in the second request by applying any of the following:

- o adding or removing whitespace, where allowed in the header field's syntax
- o combining multiple header fields with the same field name (see Section 3.2 of [Part1])
- o normalizing both header field values in a way that is known to have identical semantics, according to the header field's specification (e.g., re-ordering field values when order is not

significant; case-normalization, where values are defined to be case-insensitive)

If (after any normalization that might take place) a header field is absent from a request, it can only match another request if it is also absent there.

A Vary header field-value of "*" always fails to match, and subsequent requests to that resource can only be properly interpreted by the origin server.

The stored response with matching selecting header fields is known as the selected response.

If multiple selected responses are available, the most recent response (as determined by the Date header field) is used; see Section 4.

If no selected response is available, the cache can forward the presented request to the origin server in a conditional request; see Section 4.2.

4.4. Combining Partial Content

A response might transfer only a partial representation if the connection closed prematurely or if the request used one or more Range specifiers ([Part5]). After several such transfers, a cache might have received several ranges of the same representation. A cache MAY combine these ranges into a single stored response, and reuse that response to satisfy later requests, if they all share the same strong validator and the cache complies with the client requirements in Section 4.2 of [Part5].

When combining the new response with one or more stored responses, a cache MUST:

- o delete any Warning header fields in the stored response with warn-code 1xx (see Section 7.6);
- o retain any Warning header fields in the stored response with warn-code 2xx; and,
- o use other header fields provided in the new response, aside from Content-Range, to replace all instances of the corresponding header fields in the stored response.

5. Updating Caches with HEAD Responses

A response to the HEAD method is identical to what an equivalent request made with a GET would have been, except it lacks a body. This property of HEAD responses is used to both invalidate and update cached GET responses.

If one or more stored GET responses can be selected (as per Section 4.3) for a HEAD request, and the Content-Length, ETag or Last-Modified value of a HEAD response differs from that in a selected GET response, the cache **MUST** consider that selected response to be stale.

If the Content-Length, ETag and Last-Modified values of a HEAD response (when present) are the same as that in a selected GET response (as per Section 4.3), the cache **SHOULD** update the remaining header fields in the stored response using the following rules:

- o delete any Warning header fields in the stored response with warn-code 1xx (see Section 7.6);
- o retain any Warning header fields in the stored response with warn-code 2xx; and,
- o use other header fields provided in the response to replace all instances of the corresponding header fields in the stored response.

6. Request Methods that Invalidate

Because unsafe request methods (Section 2.1.1 of [Part2]) such as PUT, POST or DELETE have the potential for changing state on the origin server, intervening caches can use them to keep their contents up-to-date.

A cache **MUST** invalidate the effective Request URI (Section 5.5 of [Part1]) as well as the URI(s) in the Location and Content-Location response header fields (if present) when a non-error response to a request with an unsafe method is received.

However, a cache **MUST NOT** invalidate a URI from a Location or Content-Location response header field if the host part of that URI differs from the host part in the effective request URI (Section 5.5 of [Part1]). This helps prevent denial of service attacks.

A cache **MUST** invalidate the effective request URI (Section 5.5 of [Part1]) when it receives a non-error response to a request with a method whose safety is unknown.

Here, a "non-error response" is one with a 2xx (Successful) or 3xx (Redirection) status code. "Invalidate" means that the cache will either remove all stored responses related to the effective request URI, or will mark these as "invalid" and in need of a mandatory validation before they can be returned in response to a subsequent request.

Note that this does not guarantee that all appropriate responses are invalidated. For example, the request that caused the change at the origin server might not have gone through the cache where a response is stored.

7. Header Field Definitions

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

7.1. Age

The "Age" header field conveys the sender's estimate of the amount of time since the response was generated or successfully validated at the origin server. Age values are calculated as specified in Section 4.1.3.

Age = delta-seconds

Age field-values are non-negative integers, representing time in seconds (see Section 1.4.1).

The presence of an Age header field in a response implies that a response is not first-hand. However, the converse is not true, since HTTP/1.0 caches might not implement the Age header field.

7.2. Cache-Control

The "Cache-Control" header field is used to specify directives for caches along the request/response chain. Such cache directives are unidirectional in that the presence of a directive in a request does not imply that the same directive is to be given in the response.

A cache MUST obey the requirements of the Cache-Control directives defined in this section. See Section 7.2.3 for information about how Cache-Control directives defined elsewhere are handled.

Note: HTTP/1.0 caches might not implement Cache-Control and might only implement Pragma: no-cache (see Section 7.4).

A proxy, whether or not it implements a cache, MUST pass cache

directives through in forwarded messages, regardless of their significance to that application, since the directives might be applicable to all recipients along the request/response chain. It is not possible to target a directive to a specific cache.

Cache directives are identified by a token, to be compared case-insensitively, and have an optional argument, that can use both token and quoted-string syntax. For the directives defined below that define arguments, recipients ought to accept both forms, even if one is documented to be preferred. For any directive not defined by this specification, recipients **MUST** accept both forms.

Cache-Control = 1#cache-directive

cache-directive = token ["=" (token / quoted-string)]

For the cache directives defined below, no argument is defined (nor allowed) otherwise stated otherwise.

7.2.1. Request Cache-Control Directives

7.2.1.1. no-cache

The "no-cache" request directive indicates that a cache **MUST NOT** use a stored response to satisfy the request without successful validation on the origin server.

7.2.1.2. no-store

The "no-store" request directive indicates that a cache **MUST NOT** store any part of either this request or any response to it. This directive applies to both private and shared caches. "MUST NOT store" in this context means that the cache **MUST NOT** intentionally store the information in non-volatile storage, and **MUST** make a best-effort attempt to remove the information from volatile storage as promptly as possible after forwarding it.

This directive is **NOT** a reliable or sufficient mechanism for ensuring privacy. In particular, malicious or compromised caches might not recognize or obey this directive, and communications networks might be vulnerable to eavesdropping.

Note that if a request containing this directive is satisfied from a cache, the no-store request directive does not apply to the already stored response.

7.2.1.3. max-age

Argument syntax:

delta-seconds (see Section 1.4.1)

The "max-age" request directive indicates that the client is unwilling to accept a response whose age is greater than the specified number of seconds. Unless the max-stale request directive is also present, the client is not willing to accept a stale response.

Note: This directive uses the token form of the argument syntax; e.g., 'max-age=5', not 'max-age="5"'. Senders SHOULD NOT use the quoted-string form.

7.2.1.4. max-stale

Argument syntax:

delta-seconds (see Section 1.4.1)

The "max-stale" request directive indicates that the client is willing to accept a response that has exceeded its expiration time. If max-stale is assigned a value, then the client is willing to accept a response that has exceeded its expiration time by no more than the specified number of seconds. If no value is assigned to max-stale, then the client is willing to accept a stale response of any age.

Note: This directive uses the token form of the argument syntax; e.g., 'max-stale=10', not 'max-stale="10"'. Senders SHOULD NOT use the quoted-string form.

7.2.1.5. min-fresh

Argument syntax:

delta-seconds (see Section 1.4.1)

The "min-fresh" request directive indicates that the client is willing to accept a response whose freshness lifetime is no less than its current age plus the specified time in seconds. That is, the client wants a response that will still be fresh for at least the specified number of seconds.

Note: This directive uses the token form of the argument syntax; e.g., 'min-fresh=20', not 'min-fresh="20"'. Senders SHOULD NOT use

the quoted-string form.

7.2.1.6. no-transform

The "no-transform" request directive indicates that an intermediary (whether or not it implements a cache) MUST NOT change the Content-Encoding, Content-Range or Content-Type request header fields, nor the request representation.

7.2.1.7. only-if-cached

The "only-if-cached" request directive indicates that the client only wishes to obtain a stored response. If it receives this directive, a cache SHOULD either respond using a stored response that is consistent with the other constraints of the request, or respond with a 504 (Gateway Timeout) status code. If a group of caches is being operated as a unified system with good internal connectivity, a member cache MAY forward such a request within that group of caches.

7.2.2. Response Cache-Control Directives

7.2.2.1. public

The "public" response directive indicates that a response whose associated request contains an 'Authentication' header MAY be stored (see Section 3.2).

7.2.2.2. private

Argument syntax:

#field-name

The "private" response directive indicates that the response message is intended for a single user and MUST NOT be stored by a shared cache. A private cache MAY store the response.

If the private response directive specifies one or more field-names, this requirement is limited to the field-values associated with the listed response header fields. That is, a shared cache MUST NOT store the specified field-names(s), whereas it MAY store the remainder of the response message.

The field-names given are not limited to the set of standard header fields defined by this specification. Field names are case-insensitive.

Note: This usage of the word "private" only controls where the

response can be stored; it cannot ensure the privacy of the message content. Also, private response directives with field-names are often handled by implementations as if an unqualified private directive was received; i.e., the special handling for the qualified form is not widely implemented.

Note: This directive uses the quoted-string form of the argument syntax. Senders SHOULD NOT use the token form (even if quoting appears not to be needed for single-entry lists).

7.2.2.3. no-cache

Argument syntax:

#field-name

The "no-cache" response directive indicates that the response MUST NOT be used to satisfy a subsequent request without successful validation on the origin server. This allows an origin server to prevent a cache from using it to satisfy a request without contacting it, even by caches that have been configured to return stale responses.

If the no-cache response directive specifies one or more field-names, then a cache MAY use the response to satisfy a subsequent request, subject to any other restrictions on caching. However, any header fields in the response that have the field-name(s) listed MUST NOT be sent in the response to a subsequent request without successful revalidation with the origin server. This allows an origin server to prevent the re-use of certain header fields in a response, while still allowing caching of the rest of the response.

The field-names given are not limited to the set of standard header fields defined by this specification. Field names are case-insensitive.

Note: Many HTTP/1.0 caches will not recognize or obey this directive. Also, no-cache response directives with field-names are often handled by implementations as if an unqualified no-cache directive was received; i.e., the special handling for the qualified form is not widely implemented.

Note: This directive uses the quoted-string form of the argument syntax. Senders SHOULD NOT use the token form (even if quoting appears not to be needed for single-entry lists).

7.2.2.4. no-store

The "no-store" response directive indicates that a cache MUST NOT store any part of either the immediate request or response. This directive applies to both private and shared caches. "MUST NOT store" in this context means that the cache MUST NOT intentionally store the information in non-volatile storage, and MUST make a best-effort attempt to remove the information from volatile storage as promptly as possible after forwarding it.

This directive is NOT a reliable or sufficient mechanism for ensuring privacy. In particular, malicious or compromised caches might not recognize or obey this directive, and communications networks might be vulnerable to eavesdropping.

7.2.2.5. must-revalidate

The "must-revalidate" response directive indicates that once it has become stale, a cache MUST NOT use the response to satisfy subsequent requests without successful validation on the origin server.

The must-revalidate directive is necessary to support reliable operation for certain protocol features. In all circumstances a cache MUST obey the must-revalidate directive; in particular, if a cache cannot reach the origin server for any reason, it MUST generate a 504 (Gateway Timeout) response.

The must-revalidate directive ought to be used by servers if and only if failure to validate a request on the representation could result in incorrect operation, such as a silently unexecuted financial transaction.

7.2.2.6. proxy-revalidate

The "proxy-revalidate" response directive has the same meaning as the must-revalidate response directive, except that it does not apply to private caches.

7.2.2.7. max-age

Argument syntax:

delta-seconds (see Section 1.4.1)

The "max-age" response directive indicates that the response is to be considered stale after its age is greater than the specified number of seconds.

Note: This directive uses the token form of the argument syntax; e.g., 'max-age=5', not 'max-age="5"'. Senders SHOULD NOT use the quoted-string form.

7.2.2.8. s-maxage

Argument syntax:

delta-seconds (see Section 1.4.1)

The "s-maxage" response directive indicates that, in shared caches, the maximum age specified by this directive overrides the maximum age specified by either the max-age directive or the Expires header field. The s-maxage directive also implies the semantics of the proxy-revalidate response directive.

Note: This directive uses the token form of the argument syntax; e.g., 's-maxage=10', not 's-maxage="10"'. Senders SHOULD NOT use the quoted-string form.

7.2.2.9. no-transform

The "no-transform" response directive indicates that an intermediary (regardless of whether it implements a cache) MUST NOT change the Content-Encoding, Content-Range or Content-Type response header fields, nor the response representation.

7.2.3. Cache Control Extensions

The Cache-Control header field can be extended through the use of one or more cache-extension tokens, each with an optional value. Informational extensions (those that do not require a change in cache behavior) can be added without changing the semantics of other directives. Behavioral extensions are designed to work by acting as modifiers to the existing base of cache directives. Both the new directive and the standard directive are supplied, such that applications that do not understand the new directive will default to the behavior specified by the standard directive, and those that understand the new directive will recognize it as modifying the requirements associated with the standard directive. In this way, extensions to the cache-control directives can be made without requiring changes to the base protocol.

This extension mechanism depends on an HTTP cache obeying all of the cache-control directives defined for its native HTTP-version, obeying certain extensions, and ignoring all directives that it does not understand.

For example, consider a hypothetical new response directive called "community" that acts as a modifier to the private directive. We define this new directive to mean that, in addition to any private cache, any cache that is shared only by members of the community named within its value is allowed to cache the response. An origin server wishing to allow the UCI community to use an otherwise private response in their shared cache(s) could do so by including

```
Cache-Control: private, community="UCI"
```

A cache seeing this header field will act correctly even if the cache does not understand the community cache-extension, since it will also see and understand the private directive and thus default to the safe behavior.

A cache MUST ignore unrecognized cache directives; it is assumed that any cache directive likely to be unrecognized by an HTTP/1.1 cache will be combined with standard directives (or the response's default cacheability) such that the cache behavior will remain minimally correct even if the cache does not understand the extension(s).

New extension directives ought to consider defining:

- o What it means for a directive to be specified multiple times,
- o When the directive does not take an argument, what it means when an argument is present,
- o When the directive requires an argument, what it means when it is missing.

The HTTP Cache Directive Registry defines the name space for the cache directives.

A registration MUST include the following fields:

- o Cache Directive Name
- o Pointer to specification text

Values to be added to this name space require IETF Review (see [RFC5226], Section 4.1).

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

7.3. Expires

The "Expires" header field gives the date/time after which the response is considered stale. See Section 4.1 for further discussion of the freshness model.

The presence of an Expires field does not imply that the original resource will change or cease to exist at, before, or after that time.

The field-value is an absolute date and time as defined by HTTP-date in Section 5.1 of [Part2]; a sender **MUST** use the rfc1123-date format.

Expires = HTTP-date

For example

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

A cache **MUST** treat other invalid date formats, especially including the value "0", as in the past (i.e., "already expired").

Note: If a response includes a Cache-Control field with the max-age directive (see Section 7.2.2.7), that directive overrides the Expires field. Likewise, the s-maxage directive (Section 7.2.2.8) overrides the Expires header field in shared caches.

Historically, HTTP required the Expires field-value to be no more than a year in the future. While longer freshness lifetimes are no longer prohibited, extremely large values have been demonstrated to cause problems (e.g., clock overflows due to use of 32-bit integers for time values), and many caches will evict a response far sooner than that. Therefore, senders ought not produce them.

An origin server without a clock **MUST NOT** assign Expires 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).

7.4. Pragma

The "Pragma" header field allows backwards compatibility with HTTP/1.0 caches, so that clients can specify a "no-cache" request that they will understand (as Cache-Control was not defined until HTTP/1.1). When the Cache-Control header field is also present and understood in a request, Pragma is ignored.

In HTTP/1.0, Pragma was defined as an extensible field for implementation-specified directives for recipients. This specification deprecates such extensions to improve interoperability.

```
Pragma          = 1#pragma-directive
pragma-directive = "no-cache" / extension-pragma
extension-pragma = token [ "=" ( token / quoted-string ) ]
```

When the Cache-Control header field is not present in a request, the no-cache request pragma-directive **MUST** have the same effect on caches as if "Cache-Control: no-cache" were present (see Section 7.2.1).

When sending a no-cache request, a client ought to include both the pragma and cache-control directives, unless Cache-Control: no-cache is purposefully omitted to target other Cache-Control response directives at HTTP/1.1 caches. For example:

```
GET / HTTP/1.1
Host: www.example.com
Cache-Control: max-age=30
Pragma: no-cache
```

will constrain HTTP/1.1 caches to serve a response no older than 30 seconds, while precluding implementations that do not understand Cache-Control from serving a cached response.

Note: Because the meaning of "Pragma: no-cache" in responses is not specified, it does not provide a reliable replacement for "Cache-Control: no-cache" in them.

7.5. Vary

The "Vary" header field conveys the set of header fields that were used to select the representation.

Caches use this information, in part, to determine whether a stored response can be used to satisfy a given request; see Section 4.3.

In uncacheable or stale responses, the Vary field value advises the user agent about the criteria that were used to select the representation.

```
Vary = "*" / 1#field-name
```

The set of header fields named by the Vary field value is known as the selecting header fields.

A server SHOULD include a Vary header field with any cacheable response that is subject to server-driven negotiation. Doing so allows a cache to properly interpret future requests on that resource and informs the user agent about the presence of negotiation on that resource. A server MAY include a Vary header field with a non-cacheable response that is subject to server-driven negotiation, since this might provide the user agent with useful information about the dimensions over which the response varies at the time of the response.

A Vary field value of "*" signals that unspecified parameters not limited to the header fields (e.g., the network address of the client), play a role in the selection of the response representation; therefore, a cache cannot determine whether this response is appropriate. A proxy **MUST NOT** generate the "*" value.

The field-names given are not limited to the set of standard header fields defined by this specification. Field names are case-insensitive.

7.6. Warning

The "Warning" header field is used to carry additional information about the status or transformation of a message that might not be reflected in the message. This information is typically used to warn about possible incorrectness introduced by caching operations or transformations applied to the payload of the message.

Warnings can be used for other purposes, both cache-related and otherwise. The use of a warning, rather than an error status code, distinguishes these responses from true failures.

Warning header fields can in general be applied to any message, however some warn-codes are specific to caches and can only be applied to response messages.

```
Warning      = 1#warning-value
```

```
warning-value = warn-code SP warn-agent SP warn-text
               [SP warn-date]
```

```
warn-code    = 3DIGIT
warn-agent   = ( uri-host [ ":" port ] ) / pseudonym
               ; the name or pseudonym of the server adding
               ; the Warning header field, for use in debugging
warn-text    = quoted-string
warn-date    = DQUOTE HTTP-date DQUOTE
```

Multiple warnings can be attached to a response (either by the origin server or by a cache), including multiple warnings with the same code number, only differing in warn-text.

When this occurs, the user agent SHOULD inform the user of as many of them as possible, in the order that they appear in the response.

Systems that generate multiple Warning header fields are encouraged to order them with this user agent behavior in mind. New Warning header fields are added after any existing Warning header fields.

Warnings are assigned three digit warn-codes. The first digit indicates whether the Warning is required to be deleted from a stored response after validation:

- o 1xx Warnings describe the freshness or validation status of the response, and so MUST be deleted by a cache after validation. They can only be generated by a cache when validating a cached entry, and MUST NOT be generated in any other situation.
- o 2xx Warnings describe some aspect of the representation that is not rectified by a validation (for example, a lossy compression of the representation) and MUST NOT be deleted by a cache after validation, unless a full response is returned, in which case they MUST be.

If an implementation sends a message with one or more Warning header fields to a receiver whose version is HTTP/1.0 or lower, then the sender MUST include in each warning-value a warn-date that matches the Date header field in the message.

If a system receives a message with a warning-value that includes a warn-date, and that warn-date is different from the Date value in the response, then that warning-value MUST be deleted from the message before storing, forwarding, or using it. (preventing the consequences of naive caching of Warning header fields.) If all of the warning-values are deleted for this reason, the Warning header field MUST be deleted as well.

The following warn-codes are defined by this specification, each with a recommended warn-text in English, and a description of its meaning.

7.6.1. 110 Response is Stale

A cache SHOULD include this whenever the returned response is stale.

7.6.2. 111 Revalidation Failed

A cache SHOULD include this when returning a stale response because an attempt to validate the response failed, due to an inability to reach the server.

7.6.3. 112 Disconnected Operation

A cache SHOULD include this if it is intentionally disconnected from the rest of the network for a period of time.

7.6.4. 113 Heuristic Expiration

A cache SHOULD include this if it heuristically chose a freshness lifetime greater than 24 hours and the response's age is greater than 24 hours.

7.6.5. 199 Miscellaneous Warning

The warning text can include arbitrary information to be presented to a human user, or logged. A system receiving this warning MUST NOT take any automated action, besides presenting the warning to the user.

7.6.6. 214 Transformation Applied

MUST be added by a proxy if it applies any transformation to the representation, such as changing the content-coding, media-type, or modifying the representation data, unless this Warning code already appears in the response.

7.6.7. 299 Miscellaneous Persistent Warning

The warning text can include arbitrary information to be presented to a human user, or logged. A system receiving this warning MUST NOT take any automated action.

7.6.8. Warn Code Extensions

The HTTP Warn Code Registry defines the name space for warn codes.

A registration MUST include the following fields:

- o Warn Code (3 digits)
- o Short Description

- o Pointer to specification text

Values to be added to this name space require IETF Review (see [RFC5226], Section 4.1).

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

8. History Lists

User agents often have history mechanisms, such as "Back" buttons and history lists, that can be used to redisplay a representation retrieved earlier in a session.

The freshness model (Section 4.1) does not necessarily apply to history mechanisms. I.e., a history mechanism can display a previous representation even if it has expired.

This does not prohibit the history mechanism from telling the user that a view might be stale, or from honoring cache directives (e.g., Cache-Control: no-store).

9. IANA Considerations

9.1. Cache Directive Registry

The registration procedure for HTTP Cache Directives is defined by Section 7.2.3 of this document.

The HTTP Cache Directive Registry shall be created at
<<http://www.iana.org/assignments/http-cache-directives>> and be populated with the registrations below:

Cache Directive	Reference
max-age	Section 7.2.1.3, Section 7.2.2.7
max-stale	Section 7.2.1.4
min-fresh	Section 7.2.1.5
must-revalidate	Section 7.2.2.5
no-cache	Section 7.2.1.1, Section 7.2.2.3
no-store	Section 7.2.1.2, Section 7.2.2.4
no-transform	Section 7.2.1.6, Section 7.2.2.9
only-if-cached	Section 7.2.1.7
private	Section 7.2.2.2
proxy-revalidate	Section 7.2.2.6
public	Section 7.2.2.1
s-maxage	Section 7.2.2.8
stale-if-error	[RFC5861], Section 4
stale-while-revalidate	[RFC5861], Section 3

9.2. Warn Code Registry

The registration procedure for HTTP Warn Codes is defined by Section 7.6.8 of this document.

The HTTP Warn Code Registry shall be created at <http://www.iana.org/assignments/http-cache-directives> and be populated with the registrations below:

Warn Code	Short Description	Reference
110	Response is Stale	Section 7.6.1
111	Revalidation Failed	Section 7.6.2
112	Disconnected Operation	Section 7.6.3
113	Heuristic Expiration	Section 7.6.4
199	Miscellaneous Warning	Section 7.6.5
214	Transformation Applied	Section 7.6.6
299	Miscellaneous Persistent Warning	Section 7.6.7

9.3. 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
Age	http	standard	Section 7.1
Cache-Control	http	standard	Section 7.2
Expires	http	standard	Section 7.3
Pragma	http	standard	Section 7.4
Vary	http	standard	Section 7.5
Warning	http	standard	Section 7.6

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

10. Security Considerations

Caches expose additional potential vulnerabilities, since the contents of the cache represent an attractive target for malicious exploitation. Because cache contents persist after an HTTP request is complete, an attack on the cache can reveal information long after a user believes that the information has been removed from the network. Therefore, cache contents need to be protected as sensitive information.

11. Acknowledgments

See Section 9 of [Part1].

12. References

12.1. Normative References

- [Part1] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: Message Routing and Syntax", draft-ietf-httpbis-p1-messaging-20 (work in progress), July 2012.
- [Part2] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 2: Semantics and Payloads", draft-ietf-httpbis-p2-semantics-20 (work in progress), July 2012.
- [Part4] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 4: Conditional Requests", draft-ietf-httpbis-p4-conditional-20 (work in progress), July 2012.
- [Part5] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed.,

"HTTP/1.1, part 5: Range Requests",
draft-ietf-httpbis-p5-range-20 (work in progress),
July 2012.

[Part7] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed.,
"HTTP/1.1, part 7: Authentication",
draft-ietf-httpbis-p7-auth-20 (work in progress),
July 2012.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax
Specifications: ABNF", STD 68, RFC 5234, January 2008.

12.2. Informative References

[RFC1305] Mills, D., "Network Time Protocol (Version 3)
Specification, Implementation", RFC 1305, March 1992.

[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.

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

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

[RFC5861] Nottingham, M., "HTTP Cache-Control Extensions for Stale
Content", RFC 5861, April 2010.

Appendix A. Changes from RFC 2616

Make the specified age calculation algorithm less conservative.
(Section 4.1.3)

Remove requirement to consider Content-Location in successful
responses in order to determine the appropriate response to use.
(Section 4.2)

Clarify denial of service attack avoidance requirement. (Section 6)

Change ABNF productions for header fields to only define the field
value. (Section 7)

Do not mention RFC 2047 encoding and multiple languages in Warning header fields anymore, as these aspects never were implemented. (Section 7.6)

Introduce Cache Directive and Warn Code Registries. (Section 7.2.3 and Section 7.6.8)

Appendix B. Imported ABNF

The following core rules are included by reference, as defined in Appendix B.1 of [RFC5234]: 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), and VCHAR (any visible US-ASCII character).

The rules below are defined in [Part1]:

OWS	=	<OWS, defined in [Part1], Section 3.2.1>
field-name	=	<field-name, defined in [Part1], Section 3.2>
quoted-string	=	<quoted-string, defined in [Part1], Section 3.2.4>
token	=	<token, defined in [Part1], Section 3.2.4>
port	=	<port, defined in [Part1], Section 2.8>
pseudonym	=	<pseudonym, defined in [Part1], Section 6.2>
uri-host	=	<uri-host, defined in [Part1], Section 2.8>

The rules below are defined in other parts:

HTTP-date	=	<HTTP-date, defined in [Part2], Section 5.1>
-----------	---	--

Appendix C. Collected ABNF

```
Age = delta-seconds

Cache-Control = *( "," OWS ) cache-directive *( OWS "," [ OWS
  cache-directive ] )

Expires = HTTP-date

HTTP-date = <HTTP-date, defined in [Part2], Section 5.1>

OWS = <OWS, defined in [Part1], Section 3.2.1>

Pragma = *( "," OWS ) pragma-directive *( OWS "," [ OWS
  pragma-directive ] )

Vary = "*" / ( *( "," OWS ) field-name *( OWS "," [ OWS field-name ]
  ) )

Warning = *( "," OWS ) warning-value *( OWS "," [ OWS warning-value ]
  )

cache-directive = token [ "=" ( token / quoted-string ) ]

delta-seconds = 1*DIGIT

extension-pragma = token [ "=" ( token / quoted-string ) ]

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

port = <port, defined in [Part1], Section 2.8>
pragma-directive = "no-cache" / extension-pragma
pseudonym = <pseudonym, defined in [Part1], Section 6.2>

quoted-string = <quoted-string, defined in [Part1], Section 3.2.4>

token = <token, defined in [Part1], Section 3.2.4>

uri-host = <uri-host, defined in [Part1], Section 2.8>

warn-agent = ( uri-host [ ":" port ] ) / pseudonym
warn-code = 3DIGIT
warn-date = DQUOTE HTTP-date DQUOTE
warn-text = quoted-string
warning-value = warn-code SP warn-agent SP warn-text [ SP warn-date
  ]
```

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

Changes up to the first Working Group Last Call draft are summarized in <http://trac.tools.ietf.org/html/draft-ietf-httpbis-p6-cache-19#appendix-C>.

D.1. Since draft-ietf-httpbis-p6-cache-19

Closed issues:

- o <http://tools.ietf.org/wg/httpbis/trac/ticket/307>: "untangle Cache-Control ABNF"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/353>: "Multiple values in Cache-Control header fields"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/355>: "Case sensitivity of header fields in CC values"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/356>: "Spurious 'MAYs' "
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/360>: "enhance considerations for new cache control directives"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/361>: "ABNF requirements for recipients"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/368>: "note introduction of new IANA registries as normative changes"

Index

1	110 Response is Stale (warn code)	31
	111 Revalidation Failed (warn code)	32
	112 Disconnected Operation (warn code)	32
	113 Heuristic Expiration (warn code)	32
	199 Miscellaneous Warning (warn code)	32
2	214 Transformation Applied (warn code)	32
	299 Miscellaneous Persistent Warning (warn code)	32
A	age	5
	Age header field	20

C

- cache 4
- Cache Directives
 - max-age 22, 25
 - max-stale 22
 - min-fresh 22
 - must-revalidate 25
 - no-cache 21, 24
 - no-store 21, 25
 - no-transform 23, 26
 - only-if-cached 23
 - private 23
 - proxy-revalidate 25
 - public 23
 - s-maxage 26
- cache entry 7
- cache key 7
- Cache-Control header field 20
- cacheable 4

E

- Expires header field 28
- explicit expiration time 5

F

- first-hand 5
- fresh 5
- freshness lifetime 5

G

- Grammar
 - Age 20
 - Cache-Control 21
 - cache-directive 21
 - delta-seconds 7
 - Expires 28
 - extension-pragma 29
 - Pragma 29
 - pragma-directive 29
 - Vary 29
 - warn-agent 30
 - warn-code 30
 - warn-date 30
 - warn-text 30
 - Warning 30
 - warning-value 30

H

Header Fields	
Age	20
Cache-Control	20
Expires	28
Pragma	28
Vary	29
Warning	30
heuristic expiration time	5
M	
max-age	
Cache Directive	22, 25
max-stale	
Cache Directive	22
min-fresh	
Cache Directive	22
must-revalidate	
Cache Directive	25
N	
no-cache	
Cache Directive	21, 24
no-store	
Cache Directive	21, 25
no-transform	
Cache Directive	23, 26
O	
only-if-cached	
Cache Directive	23
P	
Pragma header field	28
private	
Cache Directive	23
private cache	4
proxy-revalidate	
Cache Directive	25
public	
Cache Directive	23
S	
s-maxage	
Cache Directive	26
shared cache	4
stale	5
strong validator	6

V

validator 5
 strong 6
Vary header field 29

W

Warn Codes
 110 Response is Stale 31
 111 Revalidation Failed 32
 112 Disconnected Operation 32
 113 Heuristic Expiration 32
 199 Miscellaneous Warning 32
 214 Transformation Applied 32
 299 Miscellaneous Persistent Warning 32
Warning header field 30

Authors' Addresses

Roy T. Fielding (editor)
Adobe Systems Incorporated
345 Park Ave
San Jose, CA 95110
USA

EMail: fielding@gbiv.com
URI: <http://roy.gbiv.com/>

Yves Lafon (editor)
World Wide Web Consortium
W3C / ERCIM
2004, rte des Lucioles
Sophia-Antipolis, AM 06902
France

EMail: ylafon@w3.org
URI: <http://www.raubacapeu.net/people/yves/>

Mark Nottingham (editor)
Rackspace

EMail: mnot@mnot.net
URI: <http://www.mnot.net/>

Julian F. Reschke (editor)
greenbytes GmbH
Hafenweg 16
Muenster, NW 48155
Germany

EMail: julian.reschke@greenbytes.de
URI: <http://greenbytes.de/tech/webdav/>

HTTPbis Working Group
Internet-Draft
Obsoletes: 2616 (if approved)
Updates: 2617 (if approved)
Intended status: Standards Track
Expires: January 17, 2013

R. Fielding, Ed.
Adobe
Y. Lafon, Ed.
W3C
J. Reschke, Ed.
greenbytes
July 16, 2012

HTTP/1.1, part 7: Authentication
draft-ietf-httpbis-p7-auth-20

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. This document defines the HTTP Authentication framework.

Editorial Note (To be removed by RFC Editor)

Discussion of this draft takes place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org), which is archived at <http://lists.w3.org/Archives/Public/ietf-http-wg/>.

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 D.1.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 17, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	4
1.1. Conformance and Error Handling	4
1.2. Syntax Notation	5
2. Access Authentication Framework	5
2.1. Challenge and Response	5
2.2. Protection Space (Realm)	7
2.3. Authentication Scheme Registry	8
2.3.1. Considerations for New Authentication Schemes	8
3. Status Code Definitions	9
3.1. 401 Unauthorized	9
3.2. 407 Proxy Authentication Required	10
4. Header Field Definitions	10
4.1. Authorization	10
4.2. Proxy-Authenticate	11
4.3. Proxy-Authorization	11
4.4. WWW-Authenticate	12
5. IANA Considerations	12
5.1. Authentication Scheme Registry	12
5.2. Status Code Registration	13
5.3. Header Field Registration	13
6. Security Considerations	13
6.1. Authentication Credentials and Idle Clients	13
6.2. Protection Spaces	14
7. Acknowledgments	14
8. References	15
8.1. Normative References	15
8.2. Informative References	15
Appendix A. Changes from RFCs 2616 and 2617	16
Appendix B. Imported ABNF	16
Appendix C. Collected ABNF	17
Appendix D. Change Log (to be removed by RFC Editor before publication)	17
D.1. Since draft-ietf-httpbis-p7-auth-19	17
Index	18

1. Introduction

This document defines HTTP/1.1 access control and authentication. It includes the relevant parts of RFC 2616 with only minor changes ([RFC2616]), plus the general framework for HTTP authentication, as previously defined in "HTTP Authentication: Basic and Digest Access Authentication" ([RFC2617]).

HTTP provides several OPTIONAL challenge-response authentication mechanisms which can be used by a server to challenge a client request and by a client to provide authentication information. The "basic" and "digest" authentication schemes continue to be specified in RFC 2617.

1.1. Conformance and Error Handling

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].

This specification targets conformance criteria according to the role of a participant in HTTP communication. Hence, HTTP requirements are placed on senders, recipients, clients, servers, user agents, intermediaries, origin servers, proxies, gateways, or caches, depending on what behavior is being constrained by the requirement. See Section 2 of [Part1] for definitions of these terms.

The verb "generate" is used instead of "send" where a requirement differentiates between creating a protocol element and merely forwarding a received element downstream.

An implementation is considered conformant if it complies with all of the requirements associated with the roles it partakes in HTTP. Note that SHOULD-level requirements are relevant here, unless one of the documented exceptions is applicable.

This document also uses ABNF to define valid protocol elements (Section 1.2). In addition to the prose requirements placed upon them, senders MUST NOT generate protocol elements that do not match the grammar defined by the ABNF rules for those protocol elements that are applicable to the sender's role. If a received protocol element is processed, the recipient MUST be able to parse any value that would match the ABNF rules for that protocol element, excluding only those rules not applicable to the recipient's role.

Unless noted otherwise, a recipient MAY attempt to recover a usable protocol element from an invalid construct. HTTP does not define specific error handling mechanisms except when they have a direct

impact on security, since different applications of the protocol require different error handling strategies. For example, a Web browser might wish to transparently recover from a response where the Location header field doesn't parse according to the ABNF, whereas a systems control client might consider any form of error recovery to be dangerous.

1.2. Syntax Notation

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] with the list rule extension defined in Section 1.2 of [Part1]. Appendix B describes rules imported from other documents. Appendix C shows the collected ABNF with the list rule expanded.

2. Access Authentication Framework

2.1. Challenge and Response

HTTP provides a simple challenge-response authentication mechanism that can be used by a server to challenge a client request and by a client to provide authentication information. It uses an extensible, case-insensitive token to identify the authentication scheme, followed by additional information necessary for achieving authentication via that scheme. The latter can either be a comma-separated list of parameters or a single sequence of characters capable of holding base64-encoded information.

Parameters are name-value pairs where the name is matched case-insensitively, and each parameter name **MUST** only occur once per challenge.

auth-scheme = token

auth-param = token BWS "=" BWS (token / quoted-string)

b64token = 1*(ALPHA / DIGIT /
"-" / "." / "_" / "~" / "+" / "/") * "="

The "b64token" syntax allows the 66 unreserved URI characters ([RFC3986]), plus a few others, so that it can hold a base64, base64url (URL and filename safe alphabet), base32, or base16 (hex) encoding, with or without padding, but excluding whitespace ([RFC4648]).

The 401 (Unauthorized) response message is used by an origin server to challenge the authorization of a user agent. This response **MUST** include a WWW-Authenticate header field containing at least one

challenge applicable to the requested resource.

The 407 (Proxy Authentication Required) response message is used by a proxy to challenge the authorization of a client and MUST include a Proxy-Authenticate header field containing at least one challenge applicable to the proxy for the requested resource.

```
challenge    = auth-scheme [ 1*SP ( b64token / #auth-param ) ]
```

Note: User agents will need to take special care in parsing the WWW-Authenticate and Proxy-Authenticate header field values because they can contain more than one challenge, or if more than one of each is provided, since the contents of a challenge can itself contain a comma-separated list of authentication parameters.

Note: Many clients fail to parse challenges containing unknown schemes. A workaround for this problem is to list well-supported schemes (such as "basic") first.

A user agent that wishes to authenticate itself with an origin server -- usually, but not necessarily, after receiving a 401 (Unauthorized) -- can do so by including an Authorization header field with the request.

A client that wishes to authenticate itself with a proxy -- usually, but not necessarily, after receiving a 407 (Proxy Authentication Required) -- can do so by including a Proxy-Authorization header field with the request.

Both the Authorization field value and the Proxy-Authorization field value contain the client's credentials for the realm of the resource being requested, based upon a challenge received from the server (possibly at some point in the past). When creating their values, the user agent ought to do so by selecting the challenge with what it considers to be the most secure auth-scheme that it understands, obtaining credentials from the user as appropriate.

```
credentials = auth-scheme [ 1*SP ( b64token / #auth-param ) ]
```

Upon a request for a protected resource that omits credentials, contains invalid credentials (e.g., a bad password) or partial credentials (e.g., when the authentication scheme requires more than one round trip), an origin server SHOULD return a 401 (Unauthorized) response. Such responses MUST include a WWW-Authenticate header field containing at least one (possibly new) challenge applicable to the requested resource.

Likewise, upon a request that requires authentication by proxies that omit credentials or contain invalid or partial credentials, a proxy SHOULD return a 407 (Proxy Authentication Required) response. Such responses MUST include a Proxy-Authenticate header field containing a (possibly new) challenge applicable to the proxy.

A server receiving credentials that are valid, but not adequate to gain access, ought to respond with the 403 (Forbidden) status code (Section 4.6.3 of [Part2]).

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

Proxies MUST forward the WWW-Authenticate and Authorization header fields unmodified and follow the rules found in Section 4.1.

2.2. Protection Space (Realm)

The authentication parameter realm is reserved for use by authentication schemes that wish to indicate the scope of protection.

A protection space is defined by the canonical root URI (the scheme and authority components of the effective request URI; see Section 5.5 of [Part1]) of the server being accessed, in combination with the realm value if present. These realms allow the protected resources on a server to be partitioned into a set of protection spaces, each with its own authentication scheme and/or authorization database. The realm value is a string, generally assigned by the origin server, which can have additional semantics specific to the authentication scheme. Note that there can be multiple challenges with the same auth-scheme but different realms.

The protection space determines the domain over which credentials can be automatically applied. If a prior request has been authorized, the same credentials MAY be reused for all other requests within that protection space for a period of time determined by the authentication scheme, parameters, and/or user preference. Unless otherwise defined by the authentication scheme, a single protection space cannot extend outside the scope of its server.

For historical reasons, senders MUST only use the quoted-string syntax. Recipients might have to support both token and quoted-string syntax for maximum interoperability with existing clients that have been accepting both notations for a long time.

2.3. Authentication Scheme Registry

The HTTP Authentication Scheme Registry defines the name space for the authentication schemes in challenges and credentials.

Registrations MUST include the following fields:

- o Authentication Scheme Name
- o Pointer to specification text
- o Notes (optional)

Values to be added to this name space require IETF Review (see [RFC5226], Section 4.1).

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

2.3.1. Considerations for New Authentication Schemes

There are certain aspects of the HTTP Authentication Framework that put constraints on how new authentication schemes can work:

- o HTTP authentication is presumed to be stateless: all of the information necessary to authenticate a request MUST be provided in the request, rather than be dependent on the server remembering prior requests. Authentication based on, or bound to, the underlying connection is outside the scope of this specification and inherently flawed unless steps are taken to ensure that the connection cannot be used by any party other than the authenticated user (see Section 2.4 of [Part1]).
- o The authentication parameter "realm" is reserved for defining Protection Spaces as defined in Section 2.2. New schemes MUST NOT use it in a way incompatible with that definition.
- o The "b64token" notation was introduced for compatibility with existing authentication schemes and can only be used once per challenge/credentials. New schemes thus ought to use the "auth-param" syntax instead, because otherwise future extensions will be impossible.
- o The parsing of challenges and credentials is defined by this specification, and cannot be modified by new authentication schemes. When the auth-param syntax is used, all parameters ought to support both token and quoted-string syntax, and syntactical constraints ought to be defined on the field value after parsing

(i.e., quoted-string processing). This is necessary so that recipients can use a generic parser that applies to all authentication schemes.

Note: The fact that the value syntax for the "realm" parameter is restricted to quoted-string was a bad design choice not to be repeated for new parameters.

- o Definitions of new schemes ought to define the treatment of unknown extension parameters. In general, a "must-ignore" rule is preferable over "must-understand", because otherwise it will be hard to introduce new parameters in the presence of legacy recipients. Furthermore, it's good to describe the policy for defining new parameters (such as "update the specification", or "use this registry").
- o Authentication schemes need to document whether they are usable in origin-server authentication (i.e., using WWW-Authenticate), and/or proxy authentication (i.e., using Proxy-Authenticate).
- o The credentials carried in an Authorization header field are specific to the User Agent, and therefore have the same effect on HTTP caches as the "private" Cache-Control response directive, within the scope of the request they appear in.

Therefore, new authentication schemes which choose not to carry credentials in the Authorization header field (e.g., using a newly defined header field) will need to explicitly disallow caching, by mandating the use of either Cache-Control request directives (e.g., "no-store") or response directives (e.g., "private").

3. Status Code Definitions

3.1. 401 Unauthorized

The request requires user authentication. The response MUST include a WWW-Authenticate header field (Section 4.4) containing a challenge applicable to the target resource. The client MAY repeat the request with a suitable Authorization header field (Section 4.1). If the request already included Authorization credentials, then the 401 response indicates that authorization has been refused for those credentials. If the 401 response contains the same challenge as the prior response, and the user agent has already attempted authentication at least once, then the user SHOULD be presented the representation that was given in the response, since that representation might include relevant diagnostic information.

3.2. 407 Proxy Authentication Required

This code is similar to 401 (Unauthorized), but indicates that the client ought to first authenticate itself with the proxy. The proxy MUST return a Proxy-Authenticate header field (Section 4.2) containing a challenge applicable to the proxy for the target resource. The client MAY repeat the request with a suitable Proxy-Authorization header field (Section 4.3).

4. Header Field Definitions

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

4.1. Authorization

The "Authorization" header field allows a user agent to authenticate itself with a server -- usually, but not necessarily, after receiving a 401 (Unauthorized) response. Its value consists of credentials containing information of the user agent for the realm of the resource being requested.

Authorization = credentials

If a request is authenticated and a realm specified, the same credentials SHOULD be valid for all other requests within this realm (assuming that the authentication scheme itself does not require otherwise, such as credentials that vary according to a challenge value or using synchronized clocks).

When a shared cache (see Section 1.2 of [Part6]) receives a request containing an Authorization field, it MUST NOT return the corresponding response as a reply to any other request, unless one of the following specific exceptions holds:

1. If the response includes the "s-maxage" cache-control directive, the cache MAY use that response in replying to a subsequent request. But (if the specified maximum age has passed) a proxy cache MUST first revalidate it with the origin server, using the header fields from the new request to allow the origin server to authenticate the new request. (This is the defined behavior for s-maxage.) If the response includes "s-maxage=0", the proxy MUST always revalidate it before re-using it.
2. If the response includes the "must-revalidate" cache-control directive, the cache MAY use that response in replying to a subsequent request. But if the response is stale, all caches MUST first revalidate it with the origin server, using the header

fields from the new request to allow the origin server to authenticate the new request.

3. If the response includes the "public" cache-control directive, it MAY be returned in reply to any subsequent request.

4.2. Proxy-Authenticate

The "Proxy-Authenticate" header field consists of at least one challenge that indicates the authentication scheme(s) and parameters applicable to the proxy for this effective request URI (Section 5.5 of [Part1]). It MUST be included as part of a 407 (Proxy Authentication Required) response.

Proxy-Authenticate = 1#challenge

Unlike WWW-Authenticate, the Proxy-Authenticate header field applies only to the current connection, and intermediaries SHOULD NOT forward it to downstream clients. However, an intermediate proxy might need to obtain its own credentials by requesting them from the downstream client, which in some circumstances will appear as if the proxy is forwarding the Proxy-Authenticate header field.

Note that the parsing considerations for WWW-Authenticate apply to this header field as well; see Section 4.4 for details.

4.3. Proxy-Authorization

The "Proxy-Authorization" header field allows the client to identify itself (or its user) to a proxy which requires authentication. Its value consists of credentials containing the authentication information of the user agent for the proxy and/or realm of the resource being requested.

Proxy-Authorization = credentials

Unlike Authorization, the Proxy-Authorization header field applies only to the next outbound proxy that demanded authentication using the Proxy-Authenticate field. When multiple proxies are used in a chain, the Proxy-Authorization header field is consumed by the first outbound proxy that was expecting to receive credentials. A proxy MAY relay the credentials from the client request to the next proxy if that is the mechanism by which the proxies cooperatively authenticate a given request.

4.4. WWW-Authenticate

The "WWW-Authenticate" header field consists of at least one challenge that indicates the authentication scheme(s) and parameters applicable to the effective request URI (Section 5.5 of [Part1]).

It MUST be included in 401 (Unauthorized) response messages and MAY be included in other response messages to indicate that supplying credentials (or different credentials) might affect the response.

WWW-Authenticate = 1#challenge

User agents are advised to take special care in parsing the WWW-Authenticate field value as it might contain more than one challenge, or if more than one WWW-Authenticate header field is provided, the contents of a challenge itself can contain a comma-separated list of authentication parameters.

For instance:

```
WWW-Authenticate: Newauth realm="apps", type=1,
                  title="Login to \"apps\"", Basic realm="simple"
```

This header field contains two challenges; one for the "Newauth" scheme with a realm value of "apps", and two additional parameters "type" and "title", and another one for the "Basic" scheme with a realm value of "simple".

Note: The challenge grammar production uses the list syntax as well. Therefore, a sequence of comma, whitespace, and comma can be considered both as applying to the preceding challenge, or to be an empty entry in the list of challenges. In practice, this ambiguity does not affect the semantics of the header field value and thus is harmless.

5. IANA Considerations

5.1. Authentication Scheme Registry

The registration procedure for HTTP Authentication Schemes is defined by Section 2.3 of this document.

The HTTP Method Authentication Scheme shall be created at <http://www.iana.org/assignments/http-authschemes>.

5.2. Status Code Registration

The HTTP Status Code Registry located at <http://www.iana.org/assignments/http-status-codes> shall be updated with the registrations below:

Value	Description	Reference
401	Unauthorized	Section 3.1
407	Proxy Authentication Required	Section 3.2

5.3. 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
Authorization	http	standard	Section 4.1
Proxy-Authenticate	http	standard	Section 4.2
Proxy-Authorization	http	standard	Section 4.3
WWW-Authenticate	http	standard	Section 4.4

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

6. 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.

6.1. Authentication Credentials and Idle Clients

Existing HTTP clients and user agents typically retain authentication information indefinitely. HTTP/1.1 does not provide a method for a server to direct clients to discard these cached credentials. This is a significant defect that requires further extensions to HTTP. Circumstances under which credential caching can interfere with the application's security model include but are not limited to:

- o Clients which have been idle for an extended period following which the server might wish to cause the client to reprompt the user for credentials.
- o Applications which include a session termination indication (such as a "logout" or "commit" button on a page) after which the server side of the application "knows" that there is no further reason for the client to retain the credentials.

This is currently under separate study. There are a number of work-arounds to parts of this problem, and we encourage the use of password protection in screen savers, idle time-outs, and other methods which mitigate the security problems inherent in this problem. In particular, user agents which cache credentials are encouraged to provide a readily accessible mechanism for discarding cached credentials under user control.

6.2. Protection Spaces

Authentication schemes that solely rely on the "realm" mechanism for establishing a protection space will expose credentials to all resources on a server. Clients that have successfully made authenticated requests with a resource can use the same authentication credentials for other resources on the same server. This makes it possible for a different resource to harvest authentication credentials for other resources.

This is of particular concern when a server hosts resources for multiple parties under the same canonical root URI (Section 2.2). Possible mitigation strategies include restricting direct access to authentication credentials (i.e., not making the content of the Authorization request header field available), and separating protection spaces by using a different host name for each party.

7. Acknowledgments

This specification takes over the definition of the HTTP Authentication Framework, previously defined in RFC 2617. We thank John Franks, Phillip M. Hallam-Baker, Jeffery L. Hostetler, Scott D. Lawrence, Paul J. Leach, Ari Luotonen, and Lawrence C. Stewart for their work on that specification. See Section 6 of [RFC2617] for further acknowledgements.

See Section 9 of [Part1] for the Acknowledgments related to this document revision.

8. References

8.1. Normative References

- [Part1] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: Message Routing and Syntax", draft-ietf-httpbis-p1-messaging-20 (work in progress), July 2012.
- [Part2] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 2: Semantics and Payloads", draft-ietf-httpbis-p2-semantics-20 (work in progress), July 2012.
- [Part6] Fielding, R., Ed., Lafon, Y., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1, part 6: Caching", draft-ietf-httpbis-p6-cache-20 (work in progress), July 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

8.2. Informative References

- [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.
- [RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.
- [RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

Appendix A. Changes from RFCs 2616 and 2617

The "realm" parameter isn't required anymore in general; consequently, the ABNF allows challenges without any auth parameters. (Section 2)

The "b64token" alternative to auth-param lists has been added for consistency with legacy authentication schemes such as "Basic". (Section 2)

Introduce Authentication Scheme Registry. (Section 2.3)

Change ABNF productions for header fields to only define the field value. (Section 4)

Appendix B. Imported ABNF

The following core rules are included by reference, as defined in Appendix B.1 of [RFC5234]: 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), and VCHAR (any visible US-ASCII character).

The rules below are defined in [Part1]:

BWS	= <BWS, defined in [Part1], Section 3.2.1>
OWS	= <OWS, defined in [Part1], Section 3.2.1>
quoted-string	= <quoted-string, defined in [Part1], Section 3.2.4>
token	= <token, defined in [Part1], Section 3.2.4>

Appendix C. Collected ABNF

```

Authorization = credentials

BWS = <BWS, defined in [Part1], Section 3.2.1>

OWS = <OWS, defined in [Part1], Section 3.2.1>

Proxy-Authenticate = *( "," OWS ) challenge *( OWS "," [ OWS
    challenge ] )
Proxy-Authorization = credentials

WWW-Authenticate = *( "," OWS ) challenge *( OWS "," [ OWS challenge
    ] )

auth-param = token BWS "=" BWS ( token / quoted-string )
auth-scheme = token

b64token = 1*( ALPHA / DIGIT / "-" / "." / "_" / "~" / "+" / "/" )
    * "="

challenge = auth-scheme [ 1*SP ( b64token / [ ( "," / auth-param ) *(
    OWS "," [ OWS auth-param ] ) ] ) ]
credentials = auth-scheme [ 1*SP ( b64token / [ ( "," / auth-param )
    *( OWS "," [ OWS auth-param ] ) ] ) ]

quoted-string = <quoted-string, defined in [Part1], Section 3.2.4>

token = <token, defined in [Part1], Section 3.2.4>

```

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

Changes up to the first Working Group Last Call draft are summarized in <http://trac.tools.ietf.org/html/draft-ietf-httpbis-p7-auth-19#appendix-C>.

D.1. Since draft-ietf-httpbis-p7-auth-19

Closed issues:

- o <http://tools.ietf.org/wg/httpbis/trac/ticket/348>: "Realms and scope"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/349>: "Strength"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/357>: "Authentication exchanges"

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/361>>: "ABNF requirements for recipients"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/368>>: "note introduction of new IANA registries as normative changes"

Index

- 4
 - 401 Unauthorized (status code) 9
 - 407 Proxy Authentication Required (status code) 10
- A
 - auth-param 5
 - auth-scheme 5
 - Authorization header field 10
- B
 - b64token 5
- C
 - Canonical Root URI 7
 - challenge 6
 - credentials 6
- G
 - Grammar
 - auth-param 5
 - auth-scheme 5
 - Authorization 10
 - b64token 5
 - challenge 6
 - credentials 6
 - Proxy-Authenticate 11
 - Proxy-Authorization 11
 - WWW-Authenticate 12
- H
 - Header Fields
 - Authorization 10
 - Proxy-Authenticate 11
 - Proxy-Authorization 11
 - WWW-Authenticate 12
- P
 - Protection Space 7
 - Proxy-Authenticate header field 11
 - Proxy-Authorization header field 11

R
 Realm 7

S
 Status Codes
 401 Unauthorized 9
 407 Proxy Authentication Required 10

W
 WWW-Authenticate header field 12

Authors' Addresses

Roy T. Fielding (editor)
Adobe Systems Incorporated
345 Park Ave
San Jose, CA 95110
USA

EMail: fielding@gbiv.com
URI: <http://roy.gbiv.com/>

Yves Lafon (editor)
World Wide Web Consortium
W3C / ERCIM
2004, rte des Lucioles
Sophia-Antipolis, AM 06902
France

EMail: ylafon@w3.org
URI: <http://www.raubacapeu.net/people/yves/>

Julian F. Reschke (editor)
greenbytes GmbH
Hafenweg 16
Muenster, NW 48155
Germany

EMail: julian.reschke@greenbytes.de
URI: <http://greenbytes.de/tech/webdav/>

Network Working Group
Internet-Draft
Updates: 2616 (if approved)
Intended status: Standards Track
Expires: June 19, 2015

T. Bray
Textuality
December 16, 2014

An HTTP Status Code to Report Legal Obstacles
draft-tbray-http-legally-restricted-status-05

Abstract

This document specifies a Hypertext Transfer Protocol (HTTP) status code for use when resource access is denied as a consequence of legal demands.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 19, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Requirements	2
3. 451 Unavailable For Legal Reasons	2
4. Security Considerations	3
5. IANA Considerations	3
6. Normative References	4
Appendix A. Acknowledgements	4
Author's Address	4

1. Introduction

This document specifies a Hypertext Transfer Protocol (HTTP) status code for use when a server operator has received a legal demand to deny access to a resource.

This status code may be used to provide transparency in circumstances where issues of law or public policy affect server operations. This transparency may be beneficial both to these operators and to end users.

[RFC4924] discusses the forces working against transparent operation of the Internet; these clearly include legal interventions to restrict access to content. As that document notes, and as Section 4 of [RFC4084] states, such restrictions should be made explicit.

Feedback should occur on the `ietf-http-wg@w3.org` mailing list, although this draft is NOT a work item of the IETF HTTPbis Working Group.

2. Requirements

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

3. 451 Unavailable For Legal Reasons

This status code indicates that the server is denying access to the resource as a consequence of a legal demand.

The server in question may not be an origin server. This type of legal demand typically most directly affects the operations of ISPs and search engines.

Responses using this status code SHOULD include an explanation, in the response body, of the details of the legal demand: the party

making it, the applicable legislation or regulation, and what classes of person and resource it applies to. For example:

```
HTTP/1.1 451 Unavailable For Legal Reasons
Content-Type: text/html
```

```
<html>
  <head><title>Unavailable For Legal Reasons</title></head>
  <body>
    <h1>Unavailable For Legal Reasons</h1>
    <p>This request may not be serviced in the Roman Province
      of Judea due to the Lex Julia Majestatis, which disallows
      access to resources hosted on servers deemed to be
      operated by the People's Front of Judea.</p>
  </body>
</html>
```

The use of the 451 status code implies neither the existence nor non-existence of the resource named in the request. That is to say, it is possible that if the legal demands were removed, a request for the resource still might not succeed.

Note that in many cases clients can still access the denied resource by using technical countermeasures such as a VPN or the Tor network.

4. Security Considerations

4.1. 451 Unavailable for Legal Reasons

The 451 status code is optional; clients cannot rely upon its use. It is possible that certain legal authorities may wish to avoid transparency, and not only demand the restriction of access to certain resources, but also avoid disclosing that the demand was made.

5. IANA Considerations

The HTTP Status Codes Registry should be updated with the following entries:

- o Code: 451
- o Description: Unavailable for Legal Reasons
- o Specification: [this document]

6. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4084] Klensin, J., "Terminology for Describing Internet Connectivity", BCP 104, RFC 4084, May 2005.
- [RFC4924] Aboba, B. and E. Davies, "Reflections on Internet Transparency", RFC 4924, July 2007.

Appendix A. Acknowledgements

Thanks to Terence Eden, who observed that the existing status code 403 was not really suitable for this situation, and suggested the creation of a new status code.

Thanks also to Ray Bradbury.

The author takes all responsibility for errors and omissions.

Author's Address

Tim Bray
Textuality

Email: tbray@textuality.com
URI: <http://www.tbray.org/ongoing/>