

HTTPbis Working Group
Internet-Draft
Intended status: Informational
Expires: November 3, 2011

J. Reschke
greenbytes
May 2, 2011

Initial Hypertext Transfer Protocol (HTTP)
Authentication Scheme Registrations
draft-ietf-httpbis-authscheme-registrations-01

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.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 November 3, 2011.

Copyright Notice

Copyright (c) 2011 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	3
Appendix B. Change Log (to be removed by RFC Editor before publication)	4
B.1. Since draft-ietf-httpbis-authscheme-registrations-00	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 2.1 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.
- [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-14 (work in progress), April 2011.

Appendix A. Initial Registry Contents

Authentication Scheme Name	Reference
Basic	[RFC2617], Section 2
Digest	[RFC2617], 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.

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 9, 2012

J. Reschke
greenbytes
July 8, 2011

Initial Hypertext Transfer Protocol (HTTP) Method Registrations
draft-ietf-httpbis-method-registrations-06

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 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=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.6.

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 9, 2012.

Copyright Notice

Copyright (c) 2011 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

1. Introduction

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.

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., Gettys, J.,
Mogul, J., Frystyk, H., Masinter,
L., Leach, P., Berners-Lee, T.,
Lafon, Y., Ed., and J. Reschke,
Ed., "HTTP/1.1, part 2: Message
Semantics",
draft-ietf-httpbis-p2-semantics-14
(work in progress), April 2011.

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.

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 12, 2012

R. Fielding, Ed.
Adobe
J. Gettys
Alcatel-Lucent
J. Mogul
HP
H. Frystyk
Microsoft
L. Masinter
Adobe
P. Leach
Microsoft
T. Berners-Lee
W3C/MIT
Y. Lafon, Ed.
W3C
J. Reschke, Ed.
greenbytes
July 11, 2011

HTTP/1.1, part 1: URIs, Connections, and Message Parsing
draft-ietf-httpbis-pl-messaging-15

Abstract

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

Editorial Note (To be removed by RFC Editor)

Discussion of this draft should take place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org), 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.16.

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 12, 2012.

Copyright Notice

Copyright (c) 2011 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.	Requirements	7
1.2.	Syntax Notation	7
1.2.1.	ABNF Extension: #rule	7
1.2.2.	Basic Rules	8
2.	HTTP-related architecture	10
2.1.	Client/Server Messaging	10
2.2.	Message Orientation and Buffering	12
2.3.	Connections and Transport Independence	12
2.4.	Intermediaries	13
2.5.	Caches	15
2.6.	Protocol Versioning	15
2.7.	Uniform Resource Identifiers	18
2.7.1.	http URI scheme	18
2.7.2.	https URI scheme	20
2.7.3.	http and https URI Normalization and Comparison	20
3.	Message Format	21
3.1.	Message Parsing Robustness	22
3.2.	Header Fields	22
3.3.	Message Body	25
3.4.	General Header Fields	28
4.	Request	28
4.1.	Request-Line	29
4.1.1.	Method	29
4.1.2.	request-target	29
4.2.	The Resource Identified by a Request	31
4.3.	Effective Request URI	32
5.	Response	33
5.1.	Status-Line	33
5.1.1.	Status Code and Reason Phrase	34
6.	Protocol Parameters	34
6.1.	Date/Time Formats: Full Date	34
6.2.	Transfer Codings	37
6.2.1.	Chunked Transfer Coding	38
6.2.2.	Compression Codings	40
6.2.3.	Transfer Coding Registry	41
6.3.	Product Tokens	42
6.4.	Quality Values	42
7.	Connections	42
7.1.	Persistent Connections	43
7.1.1.	Purpose	43
7.1.2.	Overall Operation	43
7.1.3.	Proxy Servers	45
7.1.4.	Practical Considerations	47
7.2.	Message Transmission Requirements	48
7.2.1.	Persistent Connections and Flow Control	48
7.2.2.	Monitoring Connections for Error Status Messages	49
7.2.3.	Use of the 100 (Continue) Status	49

7.2.4. Client Behavior if Server Prematurely Closes Connection	51
8. Miscellaneous notes that might disappear	52
8.1. Scheme aliases considered harmful	52
8.2. Use of HTTP for proxy communication	52
8.3. Interception of HTTP for access control	52
8.4. Use of HTTP by other protocols	52
8.5. Use of HTTP by media type specification	52
9. Header Field Definitions	52
9.1. Connection	52
9.2. Content-Length	54
9.3. Date	54
9.3.1. Clockless Origin Server Operation	55
9.4. Host	56
9.5. TE	57
9.6. Trailer	58
9.7. Transfer-Encoding	58
9.8. Upgrade	59
9.8.1. Upgrade Token Registry	60
9.9. Via	61
10. IANA Considerations	62
10.1. Header Field Registration	62
10.2. URI Scheme Registration	63
10.3. Internet Media Type Registrations	63
10.3.1. Internet Media Type message/http	63
10.3.2. Internet Media Type application/http	64
10.4. Transfer Coding Registry	65
10.5. Upgrade Token Registration	66
11. Security Considerations	66
11.1. Personal Information	66
11.2. Abuse of Server Log Information	67
11.3. Attacks Based On File and Path Names	67
11.4. DNS Spoofing	67
11.5. Proxies and Caching	68
11.6. Protocol Element Size Overflows	69
11.7. Denial of Service Attacks on Proxies	69
12. Acknowledgments	69
13. References	70
13.1. Normative References	70
13.2. Informative References	72
Appendix A. Tolerant Applications	74
Appendix B. HTTP Version History	75
B.1. Changes from HTTP/1.0	76
B.1.1. Multi-homed Web Servers	76
B.1.2. Keep-Alive Connections	76
B.2. Changes from RFC 2616	77
Appendix C. Collected ABNF	78
Appendix D. Change Log (to be removed by RFC Editor before	

	publication)	82
D.1.	Since RFC 2616	82
D.2.	Since draft-ietf-httpbis-pl-messaging-00	82
D.3.	Since draft-ietf-httpbis-pl-messaging-01	84
D.4.	Since draft-ietf-httpbis-pl-messaging-02	85
D.5.	Since draft-ietf-httpbis-pl-messaging-03	85
D.6.	Since draft-ietf-httpbis-pl-messaging-04	86
D.7.	Since draft-ietf-httpbis-pl-messaging-05	86
D.8.	Since draft-ietf-httpbis-pl-messaging-06	87
D.9.	Since draft-ietf-httpbis-pl-messaging-07	88
D.10.	Since draft-ietf-httpbis-pl-messaging-08	88
D.11.	Since draft-ietf-httpbis-pl-messaging-09	89
D.12.	Since draft-ietf-httpbis-pl-messaging-10	89
D.13.	Since draft-ietf-httpbis-pl-messaging-11	90
D.14.	Since draft-ietf-httpbis-pl-messaging-12	90
D.15.	Since draft-ietf-httpbis-pl-messaging-13	91
D.16.	Since draft-ietf-httpbis-pl-messaging-14	91
Index		91

1. Introduction

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

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

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

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

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

and message-forwarding intermediaries.

1.1. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

An implementation is not compliant if it fails to satisfy one or more of the "MUST" or "REQUIRED" level requirements for the protocols it implements. An implementation that satisfies all the "MUST" or "REQUIRED" level and all the "SHOULD" level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the "MUST" level requirements but not all the "SHOULD" level requirements for its protocols is said to be "conditionally compliant".

1.2. Syntax Notation

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234].

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

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

1.2.1. ABNF Extension: #rule

The #rule extension to the ABNF rules of [RFC5234] is used to improve readability.

A construct "#" is defined, similar to "*", for defining comma-delimited lists of elements. The full form is "<n>#<m>element" indicating at least <n> and at most <m> elements, each separated by a single comma (",") and optional whitespace (OWS, Section 1.2.2).

Thus,

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

and:

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

and for $n \geq 1$ and $m > 1$:

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

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

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

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

Note that empty elements do not contribute to the count of elements present, though.

For example, given these ABNF productions:

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

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

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

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

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

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

1.2.2. Basic Rules

HTTP/1.1 defines the sequence CR LF as the end-of-line marker for all protocol elements other than the message-body (see Appendix A for tolerant applications).

This specification uses three rules to denote the use of linear whitespace: OWS (optional whitespace), RWS (required whitespace), and BWS ("bad" whitespace).

The OWS rule is used where zero or more linear whitespace octets might appear. OWS SHOULD either not be produced or be produced as a single SP. Multiple OWS octets that occur within field-content SHOULD be replaced with a single SP before interpreting the field value or forwarding the message downstream.

RWS is used when at least one linear whitespace octet is required to separate field tokens. RWS SHOULD be produced as a single SP. Multiple RWS octets that occur within field-content SHOULD be replaced with a single SP before interpreting the field value or forwarding the message downstream.

BWS is used where the grammar allows optional whitespace for historical reasons but senders SHOULD NOT produce it in messages. HTTP/1.1 recipients MUST accept such bad optional whitespace and remove it before interpreting the field value or forwarding the message downstream.

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

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

```
word         = token / quoted-string

token        = 1*tchar

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

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

/ "]" / "?" / "=" / "{" / "}"

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

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

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

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

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

2. HTTP-related architecture

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

2.1. Client/Server Messaging

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

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

Most HTTP communication consists of a retrieval request (GET) for a representation of some resource identified by a URI. In the simplest case, this might be accomplished via a single bidirectional

connection (==) between the user agent (UA) and the origin server (O).

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

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

A server responds to the client's request by sending an HTTP response message (Section 5), beginning with a status line that includes the protocol version, a success or error code, and textual reason phrase, followed by MIME-like header fields containing server information, resource metadata, and payload metadata, an empty line to indicate the end of the header section, and finally the payload body (if any).

The following example illustrates a typical message exchange for a GET request on the URI "http://www.example.com/hello.txt":

client request:

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

server response:

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

Hello World!
```

2.2. Message Orientation and Buffering

Fundamentally, HTTP is a message-based protocol. Although message bodies can be chunked (Section 6.2.1) and implementations often make parts of a message available progressively, this is not required, and some widely-used implementations only make a message available when it is complete. Furthermore, while most proxies will progressively stream messages, some amount of buffering will take place, and some proxies might buffer messages to perform transformations, check content or provide other services.

Therefore, extensions to and uses of HTTP cannot rely on the availability of a partial message, or assume that messages will not be buffered. There are strategies that can be used to test for buffering in a given connection, but it should be understood that behaviors can differ across connections, and between requests and responses.

Recipients **MUST** consider every message in a connection in isolation; because HTTP is a stateless protocol, it cannot be assumed that two requests on the same connection are from the same client or share any other common attributes. In particular, intermediaries might mix requests from different clients into a single server connection. Note that some existing HTTP extensions (e.g., [RFC4559]) violate this requirement, thereby potentially causing interoperability and security problems.

2.3. Connections and Transport Independence

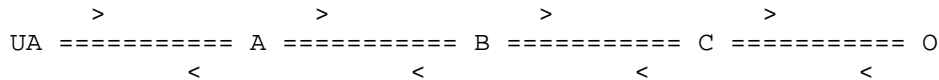
HTTP messaging is independent of the underlying transport or session-layer connection protocol(s). HTTP only presumes a reliable transport with in-order delivery of requests and the corresponding in-order delivery of responses. The mapping of HTTP request and response structures onto the data units of 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 resource's URI. For example, the "http" URI scheme (Section 2.7.1) indicates a default connection of TCP over IP, with a default TCP port of 80, but the client might be configured to use a proxy via some other connection port or protocol instead of using the defaults.

A connection might be used for multiple HTTP request/response exchanges, as defined in Section 7.1.

2.4. 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 8.2.4 of [Part2] and Section 3.6 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 comply with HTTP user agent requirements on the gateway's inbound connection and MUST implement the Connection (Section 9.1) and Via (Section 9.9) header fields for both connections.

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

In addition, there may exist network intermediaries that are not considered part of the HTTP communication but nevertheless act as filters or redirecting agents (usually violating HTTP semantics, causing security problems, and otherwise making a mess of things). Such a network intermediary, often referred to as an "interception proxy" [RFC3040], "transparent proxy" [RFC1919], or "captive portal", differs from an HTTP proxy because it has not been selected by the client. Instead, the network intermediary redirects outgoing TCP port 80 packets (and occasionally other common port traffic) to an

internal HTTP server. Interception proxies are commonly found on public network access points, as a means of enforcing account subscription prior to allowing use of non-local Internet services, and within corporate firewalls to enforce network usage policies. They are indistinguishable from a man-in-the-middle attack.

2.5. 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. Protocol Versioning

HTTP uses a "<major>.<minor>" numbering scheme to indicate versions of the protocol. This specification defines version "1.1". The protocol version as a whole indicates the sender's compliance with the set of requirements laid out in that version's corresponding specification of HTTP.

The version of an HTTP message is indicated by an HTTP-Version field in the first line of the message. HTTP-Version is case-sensitive.

```
HTTP-Version    = HTTP-Prot-Name "/" DIGIT "." DIGIT
HTTP-Prot-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 at least conditionally compliant and able to understand for future communication. The minor version advertises the sender's communication capabilities even when the sender is only using a backwards-compatible subset of the protocol, thereby letting the recipient know that more advanced features can be used in response (by servers) or in future requests (by clients).

When an HTTP/1.1 message is sent to an HTTP/1.0 recipient [RFC1945] or a recipient whose version is unknown, the HTTP/1.1 message is constructed such that it can be interpreted as a valid HTTP/1.0 message if all of the newer features are ignored. This specification places recipient-version requirements on some new features so that a compliant sender will only use compatible features until it has determined, through configuration or the receipt of a message, that the recipient supports HTTP/1.1.

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

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

Intermediaries that process HTTP messages (i.e., all intermediaries other than those acting as a tunnel) **MUST** send their own HTTP-Version in forwarded messages. In other words, they **MUST NOT** blindly forward

the first line of an HTTP message without ensuring that the protocol version matches what the intermediary understands, and is at least conditionally compliant to, for both the receiving and sending of messages. Forwarding an HTTP message without rewriting the HTTP-Version might result in communication errors when downstream recipients use the message sender's version to determine what features are safe to use for later communication with that sender.

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

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

An HTTP server SHOULD send a response version equal to the highest version for which the server is at least conditionally compliant and whose major version is less than or equal to the one received in the request. An HTTP server MUST NOT send a version for which it is not at least conditionally compliant. A server MAY send a 505 (HTTP Version Not Supported) response if it cannot send a response using the major version used in the client's request.

An HTTP server MAY send an HTTP/1.0 response to an HTTP/1.0 request if it is known or suspected that the client incorrectly implements the HTTP specification and is incapable of correctly processing later version responses, such as when a client fails to parse the version number correctly or when an intermediary is known to blindly forward the HTTP-Version even when it doesn't comply with the given minor version of the protocol. Such protocol downgrades SHOULD NOT be performed unless triggered by specific client attributes, such as when one or more of the request header fields (e.g., User-Agent) uniquely match the values sent by a client known to be in error.

The intention of HTTP's versioning design is that the major number will only be incremented if an incompatible message syntax is introduced, and that the minor number will only be incremented when changes made to the protocol have the effect of adding to the message semantics or implying additional capabilities of the sender. However, the minor version was not incremented for the changes introduced between [RFC2068] and [RFC2616], and this revision is specifically avoiding any such changes to the protocol.

2.7. Uniform Resource Identifiers

Uniform Resource Identifiers (URIs) [RFC3986] are used throughout HTTP as the means for identifying resources. URI references are used to target requests, indicate redirects, and define relationships. HTTP does not limit what a resource might be; it merely defines an interface that can be used to interact with a resource via HTTP. More information on the scope of URIs and resources can be found in [RFC3986].

This specification adopts the definitions of "URI-reference", "absolute-URI", "relative-part", "port", "host", "path-abempty", "path-absolute", "query", and "authority" from the URI generic syntax [RFC3986]. In addition, we define a partial-URI rule for protocol elements that allow a relative URI but not a fragment.

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

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

Each protocol element in HTTP that allows a URI reference will indicate in its ABNF production whether the element allows any form of reference (URI-reference), only a URI in absolute form (absolute-URI), only the path and optional query components, or some combination of the above. Unless otherwise indicated, URI references are parsed relative to the effective request URI, which defines the default base URI for references in both the request and its corresponding response.

2.7.1. 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 to the server containing the URI's identifying data as described in Section 4. If the server responds to that request with a non-interim HTTP response message, as described in Section 5, then that response is considered an authoritative answer to the client's request.

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

The URI generic syntax for authority also includes a deprecated userinfo subcomponent ([RFC3986], Section 3.2.1) for including user authentication information in the URI. Some implementations make use of the userinfo component for internal configuration of

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

2.7.2. https URI scheme

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

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

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

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

Resources made available via the "https" scheme have no shared identity with the "http" scheme even if their resource identifiers indicate the same authority (the same host listening to the same TCP port). They are distinct name spaces and are considered to be distinct origin servers. However, an extension to HTTP that is defined to apply to entire host domains, such as the Cookie protocol [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].

2.7.3. http and https URI Normalization and Comparison

Since the "http" and "https" schemes conform to the URI generic syntax, such URIs are normalized and compared according to the

algorithm defined in [RFC3986], Section 6, using the defaults described above for each scheme.

If the port is equal to the default port for a scheme, the normal form is to elide the port subcomponent. Likewise, an empty path component is equivalent to an absolute path of "/", so the normal form is to provide a path of "/" instead. The scheme and host are case-insensitive and normally provided in lowercase; all other components are compared in a case-sensitive manner. Characters other than those in the "reserved" set are equivalent to their percent-encoded octets (see [RFC3986], Section 2.1): the normal form is to not encode them.

For example, the following three URIs are equivalent:

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

3. Message Format

All HTTP/1.1 messages consist of a start-line followed by a sequence of octets in a format similar to the Internet Message Format [RFC5322]: zero or more header fields (collectively referred to as the "headers" or the "header section"), an empty line indicating the end of the header section, and an optional message-body.

An HTTP message can either be a request from client to server or a response from server to client. Syntactically, the two types of message differ only in the start-line, which is either a Request-Line (for requests) or a Status-Line (for responses), and in the algorithm for determining the length of the message-body (Section 3.3). In theory, a client could receive requests and a server could receive responses, distinguishing them by their different start-line formats, but in practice servers are implemented to only expect a request (a response is interpreted as an unknown or invalid request method) and clients are implemented to only expect a response.

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

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

result in a security vulnerability if other implementations within the request chain interpret the same message differently. Likewise, the presence of such whitespace in a response might be ignored by some clients or cause others to cease parsing.

3.1. Message Parsing Robustness

In the interest of robustness, servers SHOULD ignore at least one empty line received where a Request-Line is expected. In other words, if the server is reading the protocol stream at the beginning of a message and receives a CRLF first, it SHOULD ignore the CRLF.

Some old HTTP/1.0 client implementations send an extra CRLF after a POST request as a lame workaround for some early server applications that failed to read message-body content that was not terminated by a line-ending. An HTTP/1.1 client MUST NOT preface or follow a request with an extra CRLF. If terminating the request message-body with a line-ending is desired, then the client MUST include the terminating CRLF octets as part of the message-body length.

When a server listening only for HTTP request messages, or processing what appears from the start-line to be an HTTP request message, receives a sequence of octets that does not match the HTTP-message grammar aside from the robustness exceptions listed above, the server MUST respond with an HTTP/1.1 400 (Bad Request) response.

The normal procedure for parsing an HTTP message is to read the start-line into a structure, read each header field into a hash table by field name until the empty line, and then use the parsed data to determine if a message-body is expected. If a message-body has been indicated, then it is read as a stream until an amount of octets equal to the message-body length is read or the connection is closed. Care must be taken to parse an HTTP message as a sequence of octets in an encoding that is a superset of US-ASCII. Attempting to parse HTTP as a stream of Unicode characters in a character encoding like UTF-16 might introduce security flaws due to the differing ways that such parsers interpret invalid characters.

HTTP allows the set of defined header fields to be extended without changing the protocol version (see Section 10.1). Unrecognized header fields MUST be forwarded by a proxy unless the proxy is specifically configured to block or otherwise transform such fields. Unrecognized header fields SHOULD be ignored by other recipients.

3.2. Header Fields

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

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

No whitespace is allowed between the header field name and colon. For security reasons, any request message received containing such whitespace **MUST** be rejected with a response code of 400 (Bad Request). A proxy **MUST** remove any such whitespace from a response message before forwarding the message downstream.

A field value **MAY** be preceded by optional whitespace (OWS); a single SP is preferred. The field value does not include any leading or trailing white space: OWS occurring before the first non-whitespace octet of the field value or after the last non-whitespace octet of the field value is ignored and **SHOULD** be removed before further processing (as this does not change the meaning of the header field).

The order in which header fields with differing field names are received is not significant. However, it is "good practice" to send header fields that contain control data first, such as Host on requests and Date on responses, so that implementations can decide when not to handle a message as early as possible. A server **MUST** wait until the entire header section is received before interpreting a request message, since later header fields might include conditionals, authentication credentials, or deliberately misleading duplicate header fields that would impact request processing.

Multiple header fields with the same field name **MUST NOT** be sent in a message unless the entire field value for that header field is defined as a comma-separated list [i.e., #(values)]. Multiple header fields with the same field name can be combined into one "field-name: field-value" pair, without changing the semantics of the message, by appending each subsequent field value to the combined field value in order, separated by a comma. The order in which header fields with the same field name are received is therefore significant to the interpretation of the combined field value; a proxy **MUST NOT** change the order of these field values when forwarding a message.

Note: The "Set-Cookie" header field as implemented in practice can occur multiple times, but does not use the list syntax, and thus cannot be combined into a single line ([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.

Historically, HTTP header field values could be extended over multiple lines by preceding each extra line with at least one space or horizontal tab octet (line folding). This specification

deprecates such line folding except within the message/http media type (Section 10.3.1). HTTP/1.1 senders MUST NOT produce messages that include line folding (i.e., that contain any field-content that matches the obs-fold rule) unless the message is intended for packaging within the message/http media type. HTTP/1.1 recipients SHOULD accept line folding and replace any embedded obs-fold whitespace with a single SP prior to interpreting the field value or forwarding the message downstream.

Historically, HTTP has allowed field content with text in the ISO-8859-1 [ISO-8859-1] character encoding and supported other character sets only through use of [RFC2047] encoding. In practice, most HTTP header field values use only a subset of the US-ASCII character encoding [USASCII]. Newly defined header fields SHOULD limit their field values to US-ASCII octets. Recipients SHOULD treat other (obs-text) octets in field content as opaque data.

Comments can be included in some HTTP header fields by surrounding the comment text with parentheses. Comments are only allowed in fields containing "comment" as part of their field value definition.

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

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

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

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

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 status code if the received header field(s) would be longer than the server wishes to handle.

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

Various ad-hoc limitations on header 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.

3.3. Message Body

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

message-body = *OCTET

The message-body differs from the payload body only when a transfer-coding has been applied, as indicated by the Transfer-Encoding header field (Section 9.7). If more than one Transfer-Encoding header field is present in a message, the multiple field-values MUST be combined into one field-value, according to the algorithm defined in Section 3.2, before determining the message-body length.

When one or more transfer-codings are applied to a payload in order to form the message-body, the Transfer-Encoding header field MUST contain the list of transfer-codings applied. Transfer-Encoding is a property of the message, not of the payload, and thus MAY be added or removed by any implementation along the request/response chain under the constraints found in Section 6.2.

If a message is received that has multiple Content-Length header fields (Section 9.2) with field-values consisting of the same decimal value, or a single Content-Length header field with a field value containing a list of identical decimal values (e.g., "Content-Length: 42, 42"), indicating that duplicate Content-Length header fields have been generated or combined by an upstream message processor, then the recipient MUST 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.

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

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

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

responses MUST NOT include a message-body. All other responses do include a message-body, although the body MAY be of zero length.

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

1. Any response to a HEAD request and any response with a status code of 100-199, 204, or 304 is always terminated by the first empty line after the header fields, regardless of the header fields present in the message, and thus cannot contain a message-body.
2. If a Transfer-Encoding header field is present and the "chunked" transfer-coding (Section 6.2) is the final encoding, the message-body length is determined by reading and decoding the chunked data until the transfer-coding indicates the data is complete.

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

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

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

connection.

4. If a valid Content-Length header field is present without Transfer-Encoding, its decimal value defines the message-body length in octets. If the actual number of octets sent in the message is less than the indicated Content-Length, the recipient MUST consider the message to be incomplete and treat the connection as no longer usable. If the actual number of octets sent in the message is more than the indicated Content-Length, the recipient MUST only process the message-body up to the field value's number of octets; the remainder of the message MUST either be discarded or treated as the next message in a pipeline. For the sake of robustness, a user-agent MAY attempt to detect and correct such an error in message framing if it is parsing the response to the last request on a connection and the connection has been closed by the server.
5. If this is a request message and none of the above are true, then the message-body length is zero (no message-body is present).
6. Otherwise, this is a response message without a declared message-body length, so the message-body length is determined by the number of octets received prior to the server closing the connection.

Since there is no way to distinguish a successfully completed, close-delimited message from a partially-received message interrupted by network failure, implementations SHOULD use encoding or length-delimited messages whenever possible. The close-delimiting feature exists primarily for backwards compatibility with HTTP/1.0.

A server MAY reject a request that contains a message-body but not a Content-Length by responding with 411 (Length Required).

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

A client that sends a request containing a message-body MUST include a valid Content-Length header field if it does not know the server will handle HTTP/1.1 (or later) requests; such knowledge can be in

the form of specific user configuration or by remembering the version of a prior received response.

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

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

3.4. General Header Fields

There are a few header fields which have general applicability for both request and response messages, but which do not apply to the payload being transferred. These header fields apply only to the message being transmitted.

Header Field Name	Defined in...
Connection	Section 9.1
Date	Section 9.3
Trailer	Section 9.6
Transfer-Encoding	Section 9.7
Upgrade	Section 9.8
Via	Section 9.9

4. Request

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


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

4.1. Request-Line

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

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

4.1.1. Method

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

```
Method        = token
```

4.1.2. request-target

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

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

The four options for request-target are dependent on the nature of the request.

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

```
OPTIONS * HTTP/1.1
```

The "absolute-URI" form is REQUIRED when the request is being made to a proxy. The proxy is requested to either forward the request or service it from a valid cache, and then return the response. Note that the proxy MAY forward the request on to another proxy or directly to the server specified by the absolute-URI. In order to

avoid request loops, a proxy that forwards requests to other proxies MUST be able to recognize and exclude all of its own server names, including any aliases, local variations, and the numeric IP address. An example Request-Line would be:

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

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

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

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

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

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

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

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

For example, the request

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

would be forwarded by the final proxy as

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

after connecting to port 8001 of host "www.example.org".

The request-target is transmitted in the format specified in Section 2.7.1. If the request-target is percent-encoded ([RFC3986], Section 2.1), the origin server **MUST** decode the request-target in order to properly interpret the request. Servers **SHOULD** respond to invalid request-targets with an appropriate status code.

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

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

HTTP does not place a pre-defined limit on the length of a request-target. A server **MUST** be prepared to receive URIs of unbounded length and respond with the 414 (URI Too Long) status code if the received request-target would be longer than the server wishes to handle (see Section 8.4.15 of [Part2]).

Various ad-hoc limitations on request-target length are found in practice. It is **RECOMMENDED** that all HTTP senders and recipients support request-target lengths of 8000 or more octets.

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

4.2. The Resource Identified by a Request

The exact resource identified by an Internet request is determined by examining both the request-target and the Host header field.

An origin server that does not allow resources to differ by the requested host **MAY** ignore the Host header field value when determining the resource identified by an HTTP/1.1 request. (But see Appendix B.1.1 for other requirements on Host support in HTTP/1.1.)

An origin server that does differentiate resources based on the host requested (sometimes referred to as virtual hosts or vanity host names) MUST use the following rules for determining the requested resource on an HTTP/1.1 request:

1. If request-target is an absolute-URI, the host is part of the request-target. Any Host header field value in the request MUST be ignored.
2. If the request-target is not an absolute-URI, and the request includes a Host header field, the host is determined by the Host header field value.
3. If the host as determined by rule 1 or 2 is not a valid host on the server, the response MUST be a 400 (Bad Request) error message.

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

4.3. Effective Request URI

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

If the request-target is an absolute-URI, then the effective request URI is the request-target.

If the request-target uses the path-absolute form or the asterisk form, and the Host header field is present, then the effective request URI is constructed by concatenating

- o the scheme name: "http" if the request was received over an insecure TCP connection, or "https" when received over a SSL/TLS-secured TCP connection,
- o the octet sequence "://",
- o the authority component, as specified in the Host header field (Section 9.4), and

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

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

Otherwise, when request-target uses the authority form, the effective request URI is undefined.

Example 1: the effective request URI for the message

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

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

Example 2: the effective request URI for the message

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

(received over an SSL/TLS secured TCP connection) is "https", plus "://", plus the authority component "www.example.org", thus "https://www.example.org".

Effective request URIs are compared using the rules described in Section 2.7.3, except that empty path components MUST NOT be treated as equivalent to an absolute path of "/".

5. Response

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

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

5.1. Status-Line

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

and ending with CRLF.

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

5.1.1.1. Status Code and Reason Phrase

The Status-Code element is a 3-digit integer result code of the attempt to understand and satisfy the request. These codes are fully defined in Section 8 of [Part2]. The Reason Phrase exists for the sole purpose of providing a textual description associated with the numeric status code, out of deference to earlier Internet application protocols that were more frequently used with interactive text clients. A client SHOULD ignore the content of the Reason Phrase.

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

- o 1xx: Informational - Request received, continuing process
- o 2xx: Success - The action was successfully received, understood, and accepted
- o 3xx: Redirection - Further action must 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

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

6. Protocol Parameters

6.1. Date/Time Formats: Full Date

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

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

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

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

HTTP/1.1 clients and servers that parse a date value MUST accept all three formats (for compatibility with HTTP/1.0), though they MUST only generate the RFC 1123 format for representing HTTP-date values in header fields. See Appendix A for further information.

All HTTP date/time stamps MUST be represented in Greenwich Mean Time (GMT), without exception. For the purposes of HTTP, GMT is exactly equal to UTC (Coordinated Universal Time). This is indicated in the first two formats by the inclusion of "GMT" as the three-letter abbreviation for time zone, and MUST be assumed when reading the asctime format. HTTP-date is case sensitive and MUST NOT include additional whitespace beyond that specifically included as SP in the grammar.

HTTP-date = rfc1123-date / obs-date

Preferred format:

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

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

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

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

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

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

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

The semantics of day-name, 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.

6.2. Transfer Codings

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

```

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

```

Parameters are in the form of attribute/value pairs.

```

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

```

All transfer-coding values are case-insensitive. HTTP/1.1 uses transfer-coding values in the TE header field (Section 9.5) and in the Transfer-Encoding header field (Section 9.7).

Transfer-codings are analogous to the Content-Transfer-Encoding values of MIME, which were designed to enable safe transport of binary data over a 7-bit transport service ([RFC2045], Section 6). However, safe transport has a different focus for an 8bit-clean transfer protocol. In HTTP, the only unsafe characteristic of message-bodies is the difficulty in determining the exact message body length (Section 3.3), or the desire to encrypt data over a shared transport.

A server that receives a request message with a transfer-coding it does not understand SHOULD respond with 501 (Not Implemented) and then close the connection. A server MUST NOT send transfer-codings to an HTTP/1.0 client.

6.2.1. Chunked Transfer Coding

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

```

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

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

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

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

```

The chunk-size field is a string of hex digits indicating the size of the chunk-data in octets. The chunked encoding is ended by any chunk whose size is zero, followed by the trailer, which is terminated by an empty line.

The trailer allows the sender to include additional HTTP header fields at the end of the message. The Trailer header field can be used to indicate which header fields are included in a trailer (see Section 9.6).

A server using chunked transfer-coding in a response MUST NOT use the trailer for any header fields unless at least one of the following is true:

1. the request included a TE header field that indicates "trailers" is acceptable in the transfer-coding of the response, as described in Section 9.5; or,
2. the trailer fields consist entirely of optional metadata, and the recipient could use the message (in a manner acceptable to the server where the field originated) without receiving it. In other words, the server that generated the header (often but not always the origin server) is willing to accept the possibility that the trailer fields might be silently discarded along the path to the client.

This requirement prevents an interoperability failure when the message is being received by an HTTP/1.1 (or later) proxy and forwarded to an HTTP/1.0 recipient. It avoids a situation where compliance with the protocol would have necessitated a possibly infinite buffer on the proxy.

A process for decoding the "chunked" transfer-coding can be represented in pseudo-code as:

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

All HTTP/1.1 applications **MUST** be able to receive and decode the "chunked" transfer-coding and **MUST** ignore chunk-ext extensions they do not understand.

Since "chunked" is the only transfer-coding required to be understood by HTTP/1.1 recipients, it plays a crucial role in delimiting messages on a persistent connection. Whenever a transfer-coding is applied to a payload body in a request, the final transfer-coding applied **MUST** be "chunked". If a transfer-coding is applied to a response payload body, then either the final transfer-coding applied **MUST** be "chunked" or the message **MUST** be terminated by closing the connection. When the "chunked" transfer-coding is used, it **MUST** be the last transfer-coding applied to form the message-body. The "chunked" transfer-coding **MUST NOT** be applied more than once in a message-body.

6.2.2. Compression Codings

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

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

Note: For compatibility with previous implementations of HTTP, applications **SHOULD** consider "x-gzip" and "x-compress" to be equivalent to "gzip" and "compress" respectively.

6.2.2.1. Compress Coding

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

6.2.2.2. Deflate Coding

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

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

6.2.2.3. Gzip Coding

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

6.2.3. Transfer Coding Registry

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

Registrations MUST include the following fields:

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

Names of transfer codings MUST NOT overlap with names of content codings (Section 2.2 of [Part3]), unless the encoding transformation is identical (as it is the case for the compression codings defined in Section 6.2.2).

Values to be added to this name space require a specification (see "Specification Required" in Section 4.1 of [RFC5226]), and MUST conform to the purpose of transfer coding defined in this section.

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

6.3. Product Tokens

Product tokens are used to allow communicating applications to identify themselves by software name and version. Most fields using product tokens also allow sub-products which form a significant part of the application to be listed, separated by whitespace. By convention, the products are listed in order of their significance for identifying the application.

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

Examples:

```
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
Server: Apache/0.8.4
```

Product tokens SHOULD be short and to the point. They MUST NOT be used for advertising or other non-essential information. Although any token octet MAY appear in a product-version, this token SHOULD only be used for a version identifier (i.e., successive versions of the same product SHOULD only differ in the product-version portion of the product value).

6.4. Quality Values

Both transfer codings (TE request header field, Section 9.5) and content negotiation (Section 5 of [Part3]) use short "floating point" numbers to indicate the relative importance ("weight") of various negotiable parameters. A weight is normalized to a real number in the range 0 through 1, where 0 is the minimum and 1 the maximum value. If a parameter has a quality value of 0, then content with this parameter is "not acceptable" for the client. HTTP/1.1 applications MUST NOT generate more than three digits after the decimal point. User configuration of these values SHOULD also be limited in this fashion.

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

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

7. Connections

7.1. Persistent Connections

7.1.1. Purpose

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

Persistent HTTP connections have a number of advantages:

- 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.

7.1.2. Overall Operation

A significant difference between HTTP/1.1 and earlier versions of HTTP is that persistent connections are the default behavior of any HTTP connection. That is, unless otherwise indicated, the client SHOULD assume that the server will maintain a persistent connection,

even after error responses from the server.

Persistent connections provide a mechanism by which a client and a server can signal the close of a TCP connection. This signaling takes place using the Connection header field (Section 9.1). Once a close has been signaled, the client **MUST NOT** send any more requests on that connection.

7.1.2.1. Negotiation

An HTTP/1.1 server **MAY** assume that a HTTP/1.1 client intends to maintain a persistent connection unless a Connection header field including the connection-token "close" was sent in the request. If the server chooses to close the connection immediately after sending the response, it **SHOULD** send a Connection header field including the connection-token "close".

An HTTP/1.1 client **MAY** expect a connection to remain open, but would decide to keep it open based on whether the response from a server contains a Connection header field with the connection-token close. In case the client does not want to maintain a connection for more than that request, it **SHOULD** send a Connection header field including the connection-token close.

If either the client or the server sends the close token in the Connection header field, that request becomes the last one for the connection.

Clients and servers **SHOULD NOT** assume that a persistent connection is maintained for HTTP versions less than 1.1 unless it is explicitly signaled. See Appendix B.1.2 for more information on backward compatibility with HTTP/1.0 clients.

In order to remain persistent, all messages on the connection **MUST** have a self-defined message length (i.e., one not defined by closure of the connection), as described in Section 3.3.

7.1.2.2. Pipelining

A client that supports persistent connections **MAY** "pipeline" its requests (i.e., send multiple requests without waiting for each response). A server **MUST** send its responses to those requests in the same order that the requests were received.

Clients which assume persistent connections and pipeline immediately after connection establishment **SHOULD** be prepared to retry their connection if the first pipelined attempt fails. If a client does such a retry, it **MUST NOT** pipeline before it knows the connection is

persistent. Clients **MUST** also be prepared to resend their requests if the server closes the connection before sending all of the corresponding responses.

Clients **SHOULD NOT** pipeline requests using non-idempotent request methods or non-idempotent sequences of request methods (see Section 7.1.2 of [Part2]). Otherwise, a premature termination of the transport connection could lead to indeterminate results. A client wishing to send a non-idempotent request **SHOULD** wait to send that request until it has received the response status line for the previous request.

7.1.3. Proxy Servers

It is especially important that proxies correctly implement the properties of the Connection header field as specified in Section 9.1.

The proxy server **MUST** signal persistent connections separately with its clients and the origin servers (or other proxy servers) that it connects to. Each persistent connection applies to only one transport link.

A proxy server **MUST NOT** establish a HTTP/1.1 persistent connection with an HTTP/1.0 client (but see Section 19.7.1 of [RFC2068] for information and discussion of the problems with the Keep-Alive header field implemented by many HTTP/1.0 clients).

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

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

- 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

- o Proxy-Authenticate
- o Proxy-Authorization
- 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 9.1).

7.1.3.2. Non-modifiable Header Fields

Some features of HTTP/1.1, such as Digest Authentication, depend on the value of certain end-to-end header fields. A non-transforming proxy SHOULD NOT modify an end-to-end header field unless the definition of that header field requires or specifically allows that.

A non-transforming proxy MUST NOT modify any of the following fields in a request or response, and it MUST NOT add any of these fields if not already present:

- o Allow
- o Content-Location
- o Content-MD5
- o ETag
- o Last-Modified
- o Server

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

- o Expires

but it MAY add any of these fields if not already present. If an Expires header field is added, it MUST be given a field-value

identical to that of the Date header field in that response.

A proxy MUST NOT modify or add any of the following fields in a message that contains the no-transform cache-control directive, or in any request:

- o Content-Encoding
- o Content-Range
- o Content-Type

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

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

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

7.1.4. Practical Considerations

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

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

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

request is in progress.

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

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

Clients (including proxies) SHOULD limit the number of simultaneous connections that they maintain to a given server (including proxies).

Previous revisions of HTTP gave a specific number of connections as a ceiling, but this was found to be impractical for many applications. As a result, this specification does not mandate a particular maximum number of connections, but instead encourages clients to be conservative when opening multiple connections.

In particular, while using multiple connections avoids the "head-of-line blocking" problem (whereby a request that takes significant server-side processing and/or has a large payload can block subsequent requests on the same connection), each connection used consumes server resources (sometimes significantly), and furthermore using multiple connections can cause undesirable side effects in congested networks.

Note that servers might reject traffic that they deem abusive, including an excessive number of connections from a client.

7.2. Message Transmission Requirements

7.2.1. Persistent Connections and Flow Control

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

7.2.2. Monitoring Connections for Error Status Messages

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

7.2.3. Use of the 100 (Continue) Status

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

Requirements for HTTP/1.1 clients:

- o If a client will wait for a 100 (Continue) response before sending the request body, it MUST send an Expect header field (Section 9.2 of [Part2]) with the "100-continue" expectation.
- o A client MUST NOT send an Expect header field (Section 9.2 of [Part2]) with the "100-continue" expectation if it does not intend to send a request body.

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

Requirements for HTTP/1.1 origin servers:

- 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 is not be construed as preventing a server from defending itself against denial-of-service attacks, or from badly broken client implementations.

Requirements for HTTP/1.1 proxies:

- 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 cache recording 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 8.1 of [Part2]).

7.2.4. Client Behavior if Server Prematurely Closes Connection

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

1. Initiate a new connection to the server
2. Transmit the request-line, header fields, and the CRLF that indicates the end of header fields.
3. Initialize a variable R to the estimated round-trip time to the server (e.g., based on the time it took to establish the connection), or to a constant value of 5 seconds if the round-trip time is not available.
4. Compute $T = R * (2^{*N})$, where N is the number of previous retries of this request.
5. Wait either for an error response from the server, or for T seconds (whichever comes first)
6. If no error response is received, after T seconds transmit the body of the request.
7. If client sees that the connection is closed prematurely, repeat from step 1 until the request is accepted, an error response is received, or the user becomes impatient and terminates the retry process.

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

- o SHOULD NOT continue and

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

8. Miscellaneous notes that might disappear

8.1. Scheme aliases considered harmful

[[TBD-aliases-harmful: describe why aliases like webcal are harmful.]]

8.2. Use of HTTP for proxy communication

[[TBD-proxy-other: Configured to use HTTP to proxy HTTP or other protocols.]]

8.3. Interception of HTTP for access control

[[TBD-intercept: Interception of HTTP traffic for initiating access control.]]

8.4. Use of HTTP by other protocols

[[TBD-profiles: Profiles of HTTP defined by other protocol. Extensions of HTTP like WebDAV.]]

8.5. Use of HTTP by media type specification

[[TBD-hypertext: Instructions on composing HTTP requests via hypertext formats.]]

9. Header Field Definitions

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

9.1. Connection

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

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

```
Connection      = 1#connection-token
connection-token = token
```

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

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

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

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

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

```
Connection: close
```

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

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

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

9.2. Content-Length

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

Content-Length = 1*DIGIT

An example is

Content-Length: 3495

Implementations **SHOULD** use this field to indicate the message-body length when no transfer-coding is being applied and the payload's body length can be determined prior to being transferred. Section 3.3 describes how recipients determine the length of a message-body.

Any Content-Length greater than or equal to zero is a valid value.

Note that the use of this field in HTTP is significantly different from the corresponding definition in MIME, where it is an optional field used within the "message/external-body" content-type.

9.3. Date

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

Date = HTTP-date

An example is

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

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

1. If the response status code is 100 (Continue) or 101 (Switching Protocols), the response **MAY** include a Date header field, at the server's option.
2. If the response status code conveys a server error, e.g., 500 (Internal Server Error) or 503 (Service Unavailable), and it is inconvenient or impossible to generate a valid Date.
3. If the server does not have a clock that can provide a reasonable approximation of the current time, its responses **MUST NOT** include a Date header field. In this case, the rules in Section 9.3.1 **MUST** be followed.

A received message that does not have a Date header field **MUST** be assigned one by the recipient if the message will be cached by that recipient.

Clients can use the Date header field as well; in order to keep request messages small, they are advised not to include it when it doesn't convey any useful information (as it is usually the case for requests that do not contain a payload).

The HTTP-date sent in a Date header field **SHOULD NOT** represent a date and time subsequent to the generation of the message. It **SHOULD** represent the best available approximation of the date and time of message generation, unless the implementation has no means of generating a reasonably accurate date and time. In theory, the date ought to represent the moment just before the payload is generated. In practice, the date can be generated at any time during the message origination without affecting its semantic value.

9.3.1. Clockless Origin Server Operation

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

9.4. Host

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

Host = uri-host [":" port] ; Section 2.7.1

A client MUST send a Host header field in all HTTP/1.1 request messages. If the target resource's URI includes an authority component, then the Host field-value MUST be identical to that authority component after excluding any userinfo (Section 2.7.1). If the authority component is missing or undefined for the target resource's URI, then the Host header field MUST be sent with an empty field-value.

For example, a GET request to the origin server for <http://www.example.org/pub/WWW/> would begin with:

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

The Host header field MUST be sent in an HTTP/1.1 request even if the request-target is in the form of an absolute-URI, since this allows the Host information to be forwarded through ancient HTTP/1.0 proxies that might not have implemented Host.

When an HTTP/1.1 proxy receives a request with a request-target in the form of an absolute-URI, the proxy MUST ignore the received Host header field (if any) and instead replace it with the host information of the request-target. When a proxy forwards a request, it MUST generate the Host header field based on the received absolute-URI rather than the received Host.

Since the Host header field acts as an application-level routing mechanism, it is a frequent target for malware seeking to poison a shared cache or redirect a request to an unintended server. An interception proxy is particularly vulnerable if it relies on the Host header field value for redirecting requests to internal servers, or for use as a cache key in a shared cache, without first verifying that the intercepted connection is targeting a valid IP address for that host.

A server MUST respond with a 400 (Bad Request) status code to any HTTP/1.1 request message that lacks a Host header field and to any

request message that contains more than one Host header field or a Host header field with an invalid field-value.

See Sections 4.2 and B.1.1 for other requirements relating to Host.

9.5. TE

The "TE" header field indicates what extension transfer-codings it is willing to accept in the response, and whether or not it is willing to accept trailer fields in a chunked transfer-coding.

Its value consists of the keyword "trailers" and/or a comma-separated list of extension transfer-coding names with optional accept parameters (as described in Section 6.2).

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

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

Examples of its use are:

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

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

A server tests whether a transfer-coding is acceptable, according to a TE field, using these rules:

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

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

2. If the transfer-coding being tested is one of the transfer-codings listed in the TE field, then it is acceptable unless it is accompanied by a qvalue of 0. (As defined in Section 6.4, a qvalue of 0 means "not acceptable".)
3. If multiple transfer-codings are acceptable, then the acceptable transfer-coding with the highest non-zero qvalue is preferred. The "chunked" transfer-coding always has a qvalue of 1.

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

9.6. Trailer

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

Trailer = 1#field-name

An HTTP/1.1 message SHOULD include a Trailer header field in a message using chunked transfer-coding with a non-empty trailer. Doing so allows the recipient to know which header fields to expect in the trailer.

If no Trailer header field is present, the trailer SHOULD NOT include any header fields. See Section 6.2.1 for restrictions on the use of trailer fields in a "chunked" transfer-coding.

Message header fields listed in the Trailer header field MUST NOT include the following header fields:

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

9.7. Transfer-Encoding

The "Transfer-Encoding" header field indicates what transfer-codings (if any) have been applied to the message body. It differs from Content-Encoding (Section 2.2 of [Part3]) in that transfer-codings

are a property of the message (and therefore are removed by intermediaries), whereas content-codings are not.

Transfer-Encoding = 1#transfer-coding

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

Transfer-Encoding: chunked

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

Many older HTTP/1.0 applications do not understand the Transfer-Encoding header field.

9.8. Upgrade

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

Upgrade = 1#product

For example,

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

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

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

after changing the protocol MUST be a response to the initial HTTP request containing the Upgrade header field.

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

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

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

This specification only defines the protocol name "HTTP" for use by the family of Hypertext Transfer Protocols, as defined by the HTTP version rules of Section 2.6 and future updates to this specification. Additional tokens can be registered with IANA using the registration procedure defined below.

9.8.1. Upgrade Token Registry

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

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

1. A token, once registered, stays registered forever.
2. The registration MUST name a responsible party for the registration.
3. The registration MUST name a point of contact.
4. The registration MAY name a set of specifications associated with that token. Such specifications need not be publicly available.

5. The responsible party MAY change the registration at any time. The IANA will keep a record of all such changes, and make them available upon request.
6. The responsible party for the first registration of a "product" token MUST approve later registrations of a "version" token together with that "product" token before they can be registered.
7. If absolutely required, the IESG MAY reassign the responsibility for a token. This will normally only be used in the case when a responsible party cannot be contacted.

9.9. Via

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

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

The received-protocol indicates the protocol version of the message received by the server or client along each segment of the request/response chain. The received-protocol version is appended to the Via field value when the message is forwarded so that information about the protocol capabilities of upstream applications remains visible to all recipients.

The protocol-name is excluded if and only if it would be "HTTP". The received-by field is normally the host and optional port number of a recipient server or client that subsequently forwarded the message. However, if the real host is considered to be sensitive information, it MAY be replaced by a pseudonym. If the port is not given, it MAY be assumed to be the default port of the received-protocol.

Multiple Via field values represent each proxy or gateway that has forwarded the message. Each recipient MUST append its information such that the end result is ordered according to the sequence of

forwarding applications.

Comments MAY be used in the Via header field to identify the software of each recipient, analogous to the User-Agent and Server header fields. However, all comments in the Via field are optional and MAY be removed by any recipient prior to forwarding the message.

For example, a request message could be sent from an HTTP/1.0 user agent to an internal proxy code-named "fred", which uses HTTP/1.1 to forward the request to a public proxy at p.example.net, which completes the request by forwarding it to the origin server at www.example.com. The request received by www.example.com would then have the following Via header field:

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

A proxy or gateway used as a portal through a network firewall SHOULD NOT forward the names and ports of hosts within the firewall region unless it is explicitly enabled to do so. If not enabled, the received-by host of any host behind the firewall SHOULD be replaced by an appropriate pseudonym for that host.

For organizations that have strong privacy requirements for hiding internal structures, a proxy or gateway MAY combine an ordered subsequence of Via header field entries with identical received-protocol values into a single such entry. For example,

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

could be collapsed to

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

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

10. IANA Considerations

10.1. Header Field Registration

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

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

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

10.2. URI Scheme Registration

The entries for the "http" and "https" URI Schemes in the registry located at <http://www.iana.org/assignments/uri-schemes.html> shall be updated to point to Sections 2.7.1 and 2.7.2 of this document (see [RFC4395]).

10.3. Internet Media Type Registrations

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

10.3.1. Internet Media Type message/http

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

Type name: message

Subtype name: http

Required parameters: none

Optional parameters: version, msgtype

version: The HTTP-Version number of the enclosed message (e.g., "1.1"). If not present, the version can be determined from the first line of the body.

msgtype: The message type -- "request" or "response". If not present, the type can be determined from the first line of the body.

Encoding considerations: only "7bit", "8bit", or "binary" are permitted

Security considerations: none

Interoperability considerations: none

Published specification: This specification (see Section 10.3.1).

Applications that use this media type:

Additional information:

Magic number(s): none

File extension(s): none

Macintosh file type code(s): none

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

Intended usage: COMMON

Restrictions on usage: none

Author/Change controller: IESG

10.3.2. Internet Media Type application/http

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

Type name: application

Subtype name: http

Required parameters: none

Optional parameters: version, msgtype

version: The HTTP-Version number of the enclosed messages (e.g., "1.1"). If not present, the version can be determined from the first line of the body.

msgtype: The message type -- "request" or "response". If not present, the type can be determined from the first line of the body.

Encoding considerations: HTTP messages enclosed by this type are in "binary" format; use of an appropriate Content-Transfer-Encoding is required when transmitted via E-mail.

Security considerations: none

Interoperability considerations: none

Published specification: This specification (see Section 10.3.2).

Applications that use this media type:

Additional information:

Magic number(s): none

File extension(s): none

Macintosh file type code(s): none

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

Intended usage: COMMON

Restrictions on usage: none

Author/Change controller: IESG

10.4. Transfer Coding Registry

The registration procedure for HTTP Transfer Codings is now defined by Section 6.2.3 of this document.

The HTTP Transfer Codings Registry located at <http://www.iana.org/assignments/http-parameters> shall be updated with the registrations below:

Name	Description	Reference
chunked	Transfer in a series of chunks	Section 6.2.1
compress	UNIX "compress" program method	Section 6.2.2.1
deflate	"deflate" compression mechanism ([RFC1951]) used inside the "zlib" data format ([RFC1950])	Section 6.2.2.2
gzip	Same as GNU zip [RFC1952]	Section 6.2.2.3

10.5. Upgrade Token Registration

The registration procedure for HTTP Upgrade Tokens -- previously defined in Section 7.2 of [RFC2817] -- is now defined by Section 9.8.1 of this document.

The HTTP Status Code Registry located at <http://www.iana.org/assignments/http-upgrade-tokens/> shall be updated with the registration below:

Value	Description	Reference
HTTP	Hypertext Transfer Protocol	Section 2.6 of this specification

11. Security Considerations

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

11.1. Personal Information

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

11.2. Abuse of Server Log Information

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

11.3. Attacks Based On File and Path Names

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

11.4. DNS Spoofing

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

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

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

information reported by DNS.

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

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

11.5. Proxies and Caching

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

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

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

Users of a proxy need to be aware that proxies are no trustworthier than the people who run them; HTTP itself cannot solve this problem.

The judicious use of cryptography, when appropriate, might suffice to protect against a broad range of security and privacy attacks. Such cryptography is beyond the scope of the HTTP/1.1 specification.

11.6. 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 size limits on request-targets (Section 4.1.2) 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 8.4.15 of [Part2]) or request entities that are too large (Section 8.4 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.

11.7. Denial of Service Attacks on Proxies

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

12. Acknowledgments

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

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

Gary Adams, Harald Tveit Alvestrand, Keith Ball, Brian Behlendorf,

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

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

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

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

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

13. References

13.1. Normative References

- [ISO-8859-1] International Organization for Standardization, "Information technology -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1", ISO/IEC 8859-1:1998, 1998.
- [Part2] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 2: Message Semantics", draft-ietf-httpbis-p2-semantics-15 (work in progress), July 2011.

- [Part3] 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 3: Message Payload and Content Negotiation", draft-ietf-httpbis-p3-payload-15 (work in progress), July 2011.
- [Part6] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1, part 6: Caching", draft-ietf-httpbis-p6-cache-15 (work in progress), July 2011.
- [RFC1950] Deutsch, L. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, May 1996.
- RFC 1950 is an Informational RFC, thus it might be less stable than this specification. On the other hand, this downward reference was present since the publication of RFC 2068 in 1997 ([RFC2068]), therefore it is unlikely to cause problems in practice. See also [BCP97].
- [RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.
- RFC 1951 is an Informational RFC, thus it might be less stable than this specification. On the other hand, this downward reference was present since the publication of RFC 2068 in 1997 ([RFC2068]), therefore it is unlikely to cause problems in practice. See also [BCP97].
- [RFC1952] Deutsch, P., Gailly, J-L., Adler, M., Deutsch, L., and G. Randers-Pehrson, "GZIP file format specification version 4.3", RFC 1952, May 1996.
- RFC 1952 is an Informational RFC, thus it might be less stable than this specification. On the other hand, this downward reference was present since the publication of RFC 2068 in 1997 ([RFC2068]), therefore it is unlikely to cause problems in practice. See also [BCP97].
- [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.

13.2. Informative References

- [BCP97] Klensin, J. and S. Hartman, "Handling Normative References to Standards-Track Documents", BCP 97, RFC 4897, June 2007.
- [Kri2001] Kristol, D., "HTTP Cookies: Standards, Privacy, and Politics", ACM Transactions on Internet Technology Vol. 1, #2, November 2001, <<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>>.
- [RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.
- [RFC1900] Carpenter, B. and Y. Rekhter, "Renumbering Needs Work", RFC 1900, February 1996.
- [RFC1919] Chatel, M., "Classical versus Transparent IP Proxies", RFC 1919, March 1996.
- [RFC1945] Berners-Lee, T., Fielding, R., 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.
- [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>>.
- [Toul998] 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. Tolerant Applications

Although this document specifies the requirements for the generation of HTTP/1.1 messages, not all applications will be correct in their implementation. We therefore recommend that operational applications be tolerant of deviations whenever those deviations can be interpreted unambiguously.

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

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

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

- 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.

Appendix B. HTTP Version History

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

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

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

Since HTTP/0.9 did not support header fields in a request, there is

no mechanism for it to support name-based virtual hosts (selection of resource by inspection of the Host header field). Any server that implements name-based virtual hosts ought to disable support for HTTP/0.9. Most requests that appear to be HTTP/0.9 are, in fact, badly constructed HTTP/1.x requests wherein a buggy client failed to properly encode linear whitespace found in a URI reference and placed in the request-target.

B.1. Changes from HTTP/1.0

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

B.1.1. Multi-homed Web Servers

The requirements that clients and servers support the Host header field (Section 9.4), report an error if it is missing from an HTTP/1.1 request, and accept absolute URIs (Section 4.1.2) are among the most important changes defined by HTTP/1.1.

Older HTTP/1.0 clients assumed a one-to-one relationship of IP addresses and servers; there was no other established mechanism for distinguishing the intended server of a request than the IP address to which that request was directed. The Host header field was introduced during the development of HTTP/1.1 and, though it was quickly implemented by most HTTP/1.0 browsers, additional requirements were placed on all HTTP/1.1 requests in order to ensure complete adoption. At the time of this writing, most HTTP-based services are dependent upon the Host header field for targeting requests.

B.1.2. Keep-Alive Connections

For most implementations of HTTP/1.0, each connection is established by the client prior to the request and closed by the server after sending the response. However, some implementations implement the Keep-Alive version of persistent connections described in Section 19.7.1 of [RFC2068].

Some clients and servers might wish to be compatible with some previous implementations of persistent connections in HTTP/1.0 clients and servers. Persistent connections in HTTP/1.0 are explicitly negotiated as they are not the default behavior. HTTP/1.0 experimental implementations of persistent connections are faulty, and the new facilities in HTTP/1.1 are designed to rectify these problems. The problem was that some existing HTTP/1.0 clients might send Keep-Alive to a proxy server that doesn't understand Connection, which would then erroneously forward it to the next inbound server,

which would establish the Keep-Alive connection and result in a hung HTTP/1.0 proxy waiting for the close on the response. The result is that HTTP/1.0 clients must be prevented from using Keep-Alive when talking to proxies.

However, talking to proxies is the most important use of persistent connections, so that prohibition is clearly unacceptable. Therefore, we need some other mechanism for indicating a persistent connection is desired, which is safe to use even when talking to an old proxy that ignores Connection. Persistent connections are the default for HTTP/1.1 messages; we introduce a new keyword (Connection: close) for declaring non-persistence. See Section 9.1.

B.2. Changes from RFC 2616

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

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

Clarify that 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 2.6)

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

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

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

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

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

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

Change ABNF productions for header fields to only define the field value. (Section 9)

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

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

Appendix C. Collected ABNF

BWS = OWS

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

Connection = *("," OWS) connection-token *(OWS "," [OWS connection-token])

Content-Length = 1*DIGIT

Date = HTTP-date

GMT = %x47.4D.54 ; GMT

HTTP-Prot-Name = %x48.54.54.50 ; HTTP

HTTP-Version = HTTP-Prot-Name "/" DIGIT "." DIGIT

HTTP-date = rfc1123-date / obs-date

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

Host = uri-host [":" port]

Method = token

OWS = *([obs-fold] WSP)

RWS = 1*([obs-fold] WSP)

Reason-Phrase = *(WSP / VCHAR / obs-text)

Request = Request-Line *(header-field CRLF) CRLF [message-body]

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

Response = Status-Line *(header-field CRLF) CRLF [message-body]

Status-Code = 3DIGIT

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

TE = [("," / 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 ) product *( OWS "," [ OWS product ] )

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

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

date1 = day SP month SP year
date2 = day "-" month "-" 2DIGIT
date3 = month SP ( 2DIGIT / ( SP DIGIT ) )
day = 2DIGIT
day-name = %x4D.6F.6E ; Mon
      / %x54.75.65 ; Tue
      / %x57.65.64 ; Wed
      / %x54.68.75 ; Thu
      / %x46.72.69 ; Fri
      / %x53.61.74 ; Sat
      / %x53.75.6E ; Sun
day-name-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

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

```

```

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

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

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

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

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

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

```

```

received-by = ( uri-host [ ":" port ] ) / pseudonym
received-protocol = [ protocol-name "/" ] protocol-version
relative-part = <relative-part, defined in [RFC3986], Section 4.2>
request-target = "*" / absolute-URI / ( path-absolute [ "?" query ] )
               / authority
rfc1123-date = day-name "," SP date1 SP time-of-day SP GMT
rfc850-date = day-name-1 "," SP date2 SP time-of-day SP GMT

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

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

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

value = word

word = token / quoted-string

year = 4DIGIT

```

ABNF diagnostics:

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

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

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 WSP, 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 headers 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 headers 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 headers"
- 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-pl-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 structure"

D.11. Since draft-ietf-httpbis-pl-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 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 headers"
- 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 headers"

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 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/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"

Index

A

absolute-URI form (of request-target) 29
accelerator 14
application/http Media Type 64
asterisk form (of request-target) 29
authority form (of request-target) 30

B

browser 10

C

- cache 15
- cacheable 15
- captive portal 14
- chunked (Coding Format) 38
- client 10
- Coding Format
 - chunked 38
 - compress 41
 - deflate 41
 - gzip 41
- compress (Coding Format) 41
- connection 10
- Connection header field 52
- Content-Length header field 54

D

- Date header field 54
- deflate (Coding Format) 41
- downstream 13

E

- effective request URI 32

G

- gateway 14
- Grammar
 - absolute-URI 18
 - ALPHA 7
 - asctime-date 37
 - attribute 37
 - authority 18
 - BWS 9
 - chunk 38
 - chunk-data 38
 - chunk-ext 38
 - chunk-ext-name 38
 - chunk-ext-val 38
 - chunk-size 38
 - Chunked-Body 38
 - comment 24
 - Connection 53
 - connection-token 53
 - Content-Length 54
 - CR 7
 - CRLF 7
 - ctext 24
 - CTL 7
 - Date 54

date1 36
date2 37
date3 37
day 36
day-name 36
day-name-1 36
DIGIT 7
DQUOTE 7
field-content 23
field-name 23
field-value 23
GMT 36
header-field 23
HEXDIG 7
Host 56
hour 36
HTTP-date 35
HTTP-message 21
HTTP-Prot-Name 16
http-URI 18
HTTP-Version 16
https-URI 20
last-chunk 38
LF 7
message-body 25
Method 29
minute 36
month 36
obs-date 36
obs-text 10
OCTET 7
OWS 9
path-absolute 18
port 18
product 42
product-version 42
protocol-name 61
protocol-version 61
pseudonym 61
qdtex 10
qdtex-nf 38
query 18
quoted-cpair 24
quoted-pair 10
quoted-str-nf 38
quoted-string 10
qvalue 42
Reason-Phrase 34

received-by 61
received-protocol 61
Request 29
Request-Line 29
request-target 29
Response 33
rfc850-date 37
rfc1123-date 36
RWS 9
second 36
SP 7
special 9
Status-Code 34
Status-Line 34
t-codings 57
tchar 9
TE 57
te-ext 57
te-params 57
time-of-day 36
token 9
Trailer 58
trailer-part 38
transfer-coding 37
Transfer-Encoding 59
transfer-extension 37
transfer-parameter 37
Upgrade 59
uri-host 18
URI-reference 18
value 37
VCHAR 7
Via 61
word 9
WSP 7
year 36
gzip (Coding Format) 41

H

header field 21
Header Fields
 Connection 52
 Content-Length 54
 Date 54
 Host 56
 TE 57
 Trailer 58
 Transfer-Encoding 58

- Upgrade 59
- Via 61
- header section 21
- headers 21
- Host header field 56
- http URI scheme 18
- https URI scheme 20

I

- inbound 13
- interception proxy 14
- intermediary 13

M

- Media Type
 - application/http 64
 - message/http 63
- message 11
- message/http Media Type 63

N

- non-transforming proxy 13

O

- origin form (of request-target) 30
- origin server 10
- outbound 13

P

- proxy 13

R

- recipient 10
- request 11
- resource 18
- response 11
- reverse proxy 14

S

- sender 10
- server 10
- spider 10

T

- target resource 32
- TE header field 57
- Trailer header field 58
- Transfer-Encoding header field 58

transforming proxy 13
transparent proxy 14
tunnel 14

U

Upgrade header field 59
upstream 13
URI scheme
 http 18
 https 20
user agent 10

V

Via header field 61

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/>

Jim Gettys
Alcatel-Lucent Bell Labs
21 Oak Knoll Road
Carlisle, MA 01741
USA

EMail: jg@freedesktop.org
URI: <http://gettys.wordpress.com/>

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

EMail: JeffMogul@acm.org

Henrik Frystyk Nielsen
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052
USA

EMail: henrikn@microsoft.com

Larry Masinter
Adobe Systems Incorporated
345 Park Ave
San Jose, CA 95110
USA

EMail: LMM@acm.org
URI: <http://larry.masinter.net/>

Paul J. Leach
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052

EMail: paulle@microsoft.com

Tim Berners-Lee
World Wide Web Consortium
MIT Computer Science and Artificial Intelligence Laboratory
The Stata Center, Building 32
32 Vassar Street
Cambridge, MA 02139
USA

EMail: timbl@w3.org
URI: <http://www.w3.org/People/Berners-Lee/>

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

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

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

Phone: +49 251 2807760
Fax: +49 251 2807761
EMail: julian.reschke@greenbytes.de
URI: <http://greenbytes.de/tech/webdav/>

HTTPbis Working Group
Internet-Draft
Obsoletes: 2616 (if approved)
Updates: 2817 (if approved)
Intended status: Standards Track
Expires: January 12, 2012

R. Fielding, Ed.
Adobe
J. Gettys
Alcatel-Lucent
J. Mogul
HP
H. Frystyk
Microsoft
L. Masinter
Adobe
P. Leach
Microsoft
T. Berners-Lee
W3C/MIT
Y. Lafon, Ed.
W3C
J. Reschke, Ed.
greenbytes
July 11, 2011

HTTP/1.1, part 2: Message Semantics
draft-ietf-httpbis-p2- semantics-15

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World Wide Web global information initiative since 1990. This document is Part 2 of the seven-part specification that defines the protocol referred to as "HTTP/1.1" and, taken together, obsoletes RFC 2616. Part 2 defines the semantics of HTTP messages as expressed by request methods, request header fields, response status codes, and response header fields.

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://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 C.16.

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 12, 2012.

Copyright Notice

Copyright (c) 2011 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. Requirements	6

1.2. Syntax Notation	6
1.2.1. Core Rules	7
1.2.2. ABNF Rules defined in other Parts of the Specification	7
2. Method	7
2.1. Overview of Methods	8
2.2. Method Registry	8
2.2.1. Considerations for New Methods	8
3. Request Header Fields	9
4. Status Code and Reason Phrase	10
4.1. Overview of Status Codes	10
4.2. Status Code Registry	12
4.2.1. Considerations for New Status Codes	12
5. Response Header Fields	13
6. Representation	13
6.1. Identifying the Resource Associated with a Representation	13
7. Method Definitions	14
7.1. Safe and Idempotent Methods	14
7.1.1. Safe Methods	14
7.1.2. Idempotent Methods	15
7.2. OPTIONS	15
7.3. GET	16
7.4. HEAD	17
7.5. POST	17
7.6. PUT	18
7.7. DELETE	20
7.8. TRACE	21
7.9. CONNECT	21
7.9.1. Establishing a Tunnel with CONNECT	22
8. Status Code Definitions	23
8.1. Informational 1xx	23
8.1.1. 100 Continue	23
8.1.2. 101 Switching Protocols	23
8.2. Successful 2xx	24
8.2.1. 200 OK	24
8.2.2. 201 Created	24
8.2.3. 202 Accepted	25
8.2.4. 203 Non-Authoritative Information	25
8.2.5. 204 No Content	25
8.2.6. 205 Reset Content	26
8.2.7. 206 Partial Content	26
8.3. Redirection 3xx	26
8.3.1. 300 Multiple Choices	27
8.3.2. 301 Moved Permanently	27
8.3.3. 302 Found	28
8.3.4. 303 See Other	28
8.3.5. 304 Not Modified	29

8.3.6.	305 Use Proxy	29
8.3.7.	306 (Unused)	29
8.3.8.	307 Temporary Redirect	29
8.4.	Client Error 4xx	30
8.4.1.	400 Bad Request	30
8.4.2.	401 Unauthorized	30
8.4.3.	402 Payment Required	30
8.4.4.	403 Forbidden	30
8.4.5.	404 Not Found	31
8.4.6.	405 Method Not Allowed	31
8.4.7.	406 Not Acceptable	31
8.4.8.	407 Proxy Authentication Required	32
8.4.9.	408 Request Timeout	32
8.4.10.	409 Conflict	32
8.4.11.	410 Gone	32
8.4.12.	411 Length Required	33
8.4.13.	412 Precondition Failed	33
8.4.14.	413 Request Representation Too Large	33
8.4.15.	414 URI Too Long	33
8.4.16.	415 Unsupported Media Type	34
8.4.17.	416 Requested Range Not Satisfiable	34
8.4.18.	417 Expectation Failed	34
8.4.19.	426 Upgrade Required	34
8.5.	Server Error 5xx	34
8.5.1.	500 Internal Server Error	35
8.5.2.	501 Not Implemented	35
8.5.3.	502 Bad Gateway	35
8.5.4.	503 Service Unavailable	35
8.5.5.	504 Gateway Timeout	35
8.5.6.	505 HTTP Version Not Supported	36
9.	Header Field Definitions	36
9.1.	Allow	36
9.2.	Expect	36
9.3.	From	37
9.4.	Location	38
9.5.	Max-Forwards	39
9.6.	Referer	39
9.7.	Retry-After	40
9.8.	Server	40
9.9.	User-Agent	41
10.	IANA Considerations	42
10.1.	Method Registry	42
10.2.	Status Code Registry	42
10.3.	Header Field Registration	43
11.	Security Considerations	44
11.1.	Transfer of Sensitive Information	44
11.2.	Encoding Sensitive Information in URIs	45
11.3.	Location Headers and Spoofing	46

11.4. Security Considerations for CONNECT	46
12. Acknowledgments	46
13. References	46
13.1. Normative References	46
13.2. Informative References	47
Appendix A. Changes from RFC 2616	48
Appendix B. Collected ABNF	49
Appendix C. Change Log (to be removed by RFC Editor before publication)	51
C.1. Since RFC 2616	51
C.2. Since draft-ietf-httpbis-p2-semantics-00	51
C.3. Since draft-ietf-httpbis-p2-semantics-01	52
C.4. Since draft-ietf-httpbis-p2-semantics-02	52
C.5. Since draft-ietf-httpbis-p2-semantics-03	53
C.6. Since draft-ietf-httpbis-p2-semantics-04	53
C.7. Since draft-ietf-httpbis-p2-semantics-05	54
C.8. Since draft-ietf-httpbis-p2-semantics-06	54
C.9. Since draft-ietf-httpbis-p2-semantics-07	54
C.10. Since draft-ietf-httpbis-p2-semantics-08	55
C.11. Since draft-ietf-httpbis-p2-semantics-09	55
C.12. Since draft-ietf-httpbis-p2-semantics-10	55
C.13. Since draft-ietf-httpbis-p2-semantics-11	56
C.14. Since draft-ietf-httpbis-p2-semantics-12	56
C.15. Since draft-ietf-httpbis-p2-semantics-13	58
C.16. Since draft-ietf-httpbis-p2-semantics-14	58
Index	58

1. Introduction

This document defines HTTP/1.1 request and response semantics. Each HTTP message, as defined in [Part1], is in the form of either a request or a response. An HTTP server listens on a connection for HTTP requests and responds to each request, in the order received on that connection, with one or more HTTP response messages. This document defines the commonly agreed upon semantics of the HTTP uniform interface, the intentions defined by each request method, and the various response messages that might be expected as a result of applying that method to the target resource.

This document is currently disorganized in order to minimize the 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. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

An implementation is not compliant if it fails to satisfy one or more of the "MUST" or "REQUIRED" level requirements for the protocols it implements. An implementation that satisfies all the "MUST" or "REQUIRED" level and all the "SHOULD" level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the "MUST" level requirements but not all the "SHOULD" level requirements for its protocols is said to be "conditionally compliant".

1.2. Syntax Notation

This specification uses the ABNF syntax defined in Section 1.2 of [Part1] (which extends the syntax defined in [RFC5234] with a list rule). Appendix B shows the collected ABNF, with the list rule expanded.

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

sequence of data), SP (space), VCHAR (any visible USASCII character), and WSP (whitespace).

1.2.1. Core Rules

The core rules below are defined in Section 1.2.2 of [Part1]:

```
quoted-string = <quoted-string, defined in [Part1], Section 1.2.2>
token         = <token, defined in [Part1], Section 1.2.2>
OWS          = <OWS, defined in [Part1], Section 1.2.2>
RWS          = <RWS, defined in [Part1], Section 1.2.2>
obs-text     = <obs-text, defined in [Part1], Section 1.2.2>
```

1.2.2. ABNF Rules defined in other Parts of the Specification

The ABNF rules below are defined in other parts:

```
absolute-URI  = <absolute-URI, defined in [Part1], Section 2.7>
comment       = <comment, defined in [Part1], Section 3.2>
HTTP-date     = <HTTP-date, defined in [Part1], Section 6.1>
partial-URI   = <partial-URI, defined in [Part1], Section 2.7>
product       = <product, defined in [Part1], Section 6.3>
URI-reference = <URI-reference, defined in [Part1], Section 2.7>
```

2. Method

The Method token indicates the request method to be performed on the target resource (Section 4.3 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 9.1). 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 7.

2.1. Overview of Methods

The methods listed below are defined in Section 7.

Method Name	Defined in...
OPTIONS	Section 7.2
GET	Section 7.3
HEAD	Section 7.4
POST	Section 7.5
PUT	Section 7.6
DELETE	Section 7.7
TRACE	Section 7.8
CONNECT	Section 7.9

Note that this list is not exhaustive -- it does not include request methods defined in other specifications.

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 7.1.1)
- o Pointer to specification text

Values to be added to this name space are subject to IETF review ([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 may 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 7.1.1), what semantics (if any) the request body has, and whether they are idempotent (Section 7.1.2). They also need to state whether they can be cached ([Part6]); in particular what conditions a cache may store the response, and under what conditions such a stored response may be used to satisfy a subsequent request.

3. 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 6.1 of [Part3]
Accept-Charset	Section 6.2 of [Part3]
Accept-Encoding	Section 6.3 of [Part3]
Accept-Language	Section 6.4 of [Part3]
Authorization	Section 4.1 of [Part7]
Expect	Section 9.2
From	Section 9.3
Host	Section 9.4 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 9.5
Proxy-Authorization	Section 4.3 of [Part7]
Range	Section 5.4 of [Part5]
Referer	Section 9.6
TE	Section 9.5 of [Part1]

User-Agent	Section 9.9	
+-----+-----+		

4. Status Code and Reason Phrase

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

The Reason-Phrase is intended to give a short textual description of the Status-Code and is intended for a human user. The client does not need to examine or display the Reason-Phrase.

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

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.

4.1. Overview of Status Codes

The status codes listed below are defined in Section 8 of 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 8.1.1
101	Switching Protocols	Section 8.1.2
200	OK	Section 8.2.1
201	Created	Section 8.2.2
202	Accepted	Section 8.2.3
203	Non-Authoritative Information	Section 8.2.4
204	No Content	Section 8.2.5
205	Reset Content	Section 8.2.6
206	Partial Content	Section 3.1 of [Part5]
300	Multiple Choices	Section 8.3.1
301	Moved Permanently	Section 8.3.2
302	Found	Section 8.3.3
303	See Other	Section 8.3.4
304	Not Modified	Section 4.1 of [Part4]
305	Use Proxy	Section 8.3.6
307	Temporary Redirect	Section 8.3.8
400	Bad Request	Section 8.4.1
401	Unauthorized	Section 3.1 of [Part7]
402	Payment Required	Section 8.4.3
403	Forbidden	Section 8.4.4
404	Not Found	Section 8.4.5
405	Method Not Allowed	Section 8.4.6
406	Not Acceptable	Section 8.4.7
407	Proxy Authentication Required	Section 3.2 of [Part7]
408	Request Time-out	Section 8.4.9
409	Conflict	Section 8.4.10
410	Gone	Section 8.4.11
411	Length Required	Section 8.4.12
412	Precondition Failed	Section 4.2 of [Part4]
413	Request Representation Too Large	Section 8.4.14
414	URI Too Long	Section 8.4.15
415	Unsupported Media Type	Section 8.4.16
416	Requested range not satisfiable	Section 3.2 of [Part5]
417	Expectation Failed	Section 8.4.18
426	Upgrade Required	Section 8.4.19
500	Internal Server Error	Section 8.5.1
501	Not Implemented	Section 8.5.2

502	Bad Gateway	Section 8.5.3	
503	Service Unavailable	Section 8.5.4	
504	Gateway Time-out	Section 8.5.5	
505	HTTP Version not supported	Section 8.5.6	
+-----+-----+-----+-----+			

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 are subject to IETF review ([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 headers and/or method(s)), along with any interactions with response headers (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 8. 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(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 6.1; by default, it is anonymous).

5. 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 4.3 of [Part1]).

Header Field Name	Defined in...
Accept-Ranges	Section 5.1 of [Part5]
Age	Section 3.1 of [Part6]
Allow	Section 9.1
ETag	Section 2.2 of [Part4]
Location	Section 9.4
Proxy-Authenticate	Section 4.2 of [Part7]
Retry-After	Section 9.7
Server	Section 9.8
Vary	Section 3.5 of [Part6]
WWW-Authenticate	Section 4.4 of [Part7]

6. Representation

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. HTTP representations are defined in [Part3].

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.

6.1. 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 4.3 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 or 203 and the request method was GET, the response payload is a representation of the target resource.
2. If the response status code is 204, 206, or 304 and the request method was GET or HEAD, the response payload is a partial representation of the target resource (see Section 2.8 of [Part6]).
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. Method Definitions

The set of common request methods for HTTP/1.1 is defined below. Although this set can be expanded, additional methods cannot be assumed to share the same semantics for separately extended clients and servers.

7.1. Safe and Idempotent Methods

7.1.1. Safe Methods

Implementors 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.

7.1.2. 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.

7.2. 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 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 compliance (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 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 9.5). If no Max-Forwards field is present in the request, then the forwarded request MUST NOT include a Max-Forwards field.

7.3. 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. 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. 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.

7.4. 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; see [Part6]. It also MAY be used to update a previously cached representation from that resource; if the new field values indicate that the cached representation differs from the current representation (as would be indicated by a change in Content-Length, Content-MD5, ETag or Last-Modified), then the cache MUST treat the cache entry as stale.

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.

7.5. 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 9.4).

Responses to POST requests are only cacheable when they include explicit freshness information (see Section 2.3.1 of [Part6]). A cached POST response with a Content-Location header field (see Section 6.7 of [Part3]) 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 resource.

7.6. 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 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, 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 2.5 of [Part6]).

7.7. 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 an 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 2.5 of [Part6]).

7.8. 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 9.5). 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 9.9 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 10.3.1 of [Part1]) and contain a message-body that encloses a copy of the entire request message. Responses to the TRACE method are not cacheable.

7.9. CONNECT

The CONNECT method requests that the proxy establish a tunnel to the request-target and then 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 4.1.2 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
```

Other HTTP mechanisms can be used normally with the CONNECT method -- except end-to-end protocol Upgrade requests, since the tunnel must be established first.

For example, 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=
```

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

Like any other pipelined HTTP/1.1 request, data to be tunnel may be sent immediately after the blank line. The usual caveats also apply: data may be discarded if the eventual response is negative, and the connection may be reset with no response if more than one TCP segment is outstanding.

7.9.1. Establishing a Tunnel with CONNECT

Any successful (2xx) 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.

It may 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.

An origin server which receives a CONNECT request for itself MAY respond with a 2xx status code to indicate that a connection is established.

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.

8. Status Code Definitions

Each Status-Code is described below, including any metadata required in the response.

8.1. Informational 1xx

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 1xx status codes, servers **MUST NOT** send a 1xx response to an HTTP/1.0 client except under experimental conditions.

A client **MUST** be prepared to accept one or more 1xx status responses prior to a regular response, even if the client does not expect a 100 (Continue) status message. Unexpected 1xx status responses **MAY** be ignored by a user agent.

Proxies **MUST** forward 1xx responses, unless the connection between the proxy and its client has been closed, or unless the proxy itself requested the generation of the 1xx response. (For example, if a proxy adds a "Expect: 100-continue" field when it forwards a request, then it need not forward the corresponding 100 (Continue) response(s).)

8.1.1. 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 7.2.3 of [Part1] for detailed discussion of the use and handling of this status code.

8.1.2. 101 Switching Protocols

The server understands and is willing to comply with the client's request, via the Upgrade message header field (Section 9.8 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.

8.2. Successful 2xx

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

8.2.1. 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 2.3.1.1 of [Part6]) to determine freshness for 200 responses.

8.2.2. 201 Created

The request has been fulfilled and has resulted in a new resource being created. The newly created resource can be referenced by the URI(s) returned in the payload of the response, with the most specific URI for the resource given by a Location header field. The response SHOULD include a payload containing a list of resource characteristics and location(s) from which the user or user agent can choose the one most appropriate. The payload format is specified by the media type given in the Content-Type header field. The origin server MUST create the resource 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 just created (see Section 2.2 of [Part4]).

8.2.3. 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.

8.2.4. 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 behaviour of transforming intermediaries is controlled by the no-transform Cache-Control directive (Section 3.2 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 3.6 of [Part6]) is appropriate.

Caches MAY use a heuristic (see Section 2.3.1.1 of [Part6]) to determine freshness for 203 responses.

8.2.5. 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.

8.2.6. 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].

8.2.7. 206 Partial Content

The server has fulfilled the partial GET request for the resource and the enclosed payload is a partial representation as defined in Section 3.1 of [Part5].

Caches MAY use a heuristic (see Section 2.3.1.1 of [Part6]) to determine freshness for 206 responses.

8.3. Redirection 3xx

This class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request. The action required 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 7.1.1. A client SHOULD detect infinite redirection loops, since such loops generate network traffic for each redirection.

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.

8.3.1. 300 Multiple Choices

The target resource has more than one representation, each with its own specific location, and agent-driven negotiation information (Section 5 of [Part3]) 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. The data format is specified by the media type given in the Content-Type header field. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice MAY be performed automatically. 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 2.3.1.1 of [Part6]) to determine freshness for 300 responses.

8.3.2. 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 2.3.1.1 of [Part6]) to determine freshness for 301 responses.

The new permanent URI SHOULD be given by the Location field in the response. Unless the request method was HEAD, the representation of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s).

If the 301 status code is received in response to a request method that is known to be "safe", as defined in Section 7.1.1, then the

request MAY be automatically redirected by the user agent without confirmation. Otherwise, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

Note: When automatically redirecting a POST request after receiving a 301 status code, some existing HTTP/1.0 user agents will erroneously change it into a GET request.

8.3.3. 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. Unless the request method was HEAD, the representation of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s).

If the 302 status code is received in response to a request method that is known to be "safe", as defined in Section 7.1.1, then the request MAY be automatically redirected by the user agent without confirmation. Otherwise, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

Note: HTTP/1.0 ([RFC1945], Section 9.3) and the first version of HTTP/1.1 ([RFC2068], Section 10.3.3) specify that the client is not allowed to change the method on the redirected request. However, most existing user agent implementations treat 302 as if it were a 303 response, performing a GET on the Location field-value regardless of the original request method. Therefore, a previous version of this specification ([RFC2616], Section 10.3.3) has added the status codes 303 and 307 for servers that wish to make unambiguously clear which kind of reaction is expected of the client.

8.3.4. 303 See Other

The server directs the user agent to a different resource, indicated by a URI in the Location header field, that provides an indirect response to the original request. The user agent MAY perform a GET request on the URI in the Location field in order to obtain a representation corresponding to the response, be redirected again, or end with an error status. The Location URI is not a substitute reference for the effective request URI.

The 303 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.

8.3.5. 304 Not Modified

The response to the request has not been modified since the conditions indicated by the client's conditional GET request, as defined in Section 4.1 of [Part4].

8.3.6. 305 Use Proxy

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

8.3.7. 306 (Unused)

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

8.3.8. 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. Unless the request method was HEAD, the representation of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s), since many pre-HTTP/1.1 user agents do not understand the 307 status code. Therefore, the note SHOULD contain

the information necessary for a user to repeat the original request on the new URI.

If the 307 status code is received in response to a request method that is known to be "safe", as defined in Section 7.1.1, then the request MAY be automatically redirected by the user agent without confirmation. Otherwise, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

8.4. 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.

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.

8.4.1. 400 Bad Request

The request could not be understood by the server due to malformed syntax. The client SHOULD NOT repeat the request without modifications.

8.4.2. 401 Unauthorized

The request requires user authentication (see Section 3.1 of [Part7]).

8.4.3. 402 Payment Required

This code is reserved for future use.

8.4.4. 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.

8.4.5. 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.

8.4.6. 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.

8.4.7. 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 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. The data format is specified by the media type given in the Content-Type header field. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice MAY be performed automatically. 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.

8.4.8. 407 Proxy Authentication Required

This code is similar to 401 (Unauthorized), but indicates that the client must first authenticate itself with the proxy (see Section 3.2 of [Part7]).

8.4.9. 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.

8.4.10. 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 in a format defined by the response Content-Type.

8.4.11. 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 2.3.1.1 of [Part6]) to determine freshness for 410 responses.

8.4.12. 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.

8.4.13. 412 Precondition Failed

The precondition given in one or more of the header fields evaluated to false when it was tested on the server, as defined in Section 4.2 of [Part4].

8.4.14. 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.

8.4.15. 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 effective request URI.

8.4.16. 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.

8.4.17. 416 Requested Range Not Satisfiable

The request included a Range header field (Section 5.4 of [Part5]) and none of the range-specifier values in this field overlap the current extent of the selected resource. See Section 3.2 of [Part5].

8.4.18. 417 Expectation Failed

The expectation given in an Expect header field (see Section 9.2) 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.

8.4.19. 426 Upgrade Required

The request can not be completed without a prior protocol upgrade. This response MUST include an Upgrade header field (Section 9.8 of [Part1]) specifying the required protocols.

Example:

```
HTTP/1.1 426 Upgrade Required
Upgrade: HTTP/2.0
Connection: Upgrade
```

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 may be available to the user.

8.5. 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.

8.5.1. 500 Internal Server Error

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

8.5.2. 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.

8.5.3. 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.

8.5.4. 503 Service Unavailable

The server is currently unable or unwilling to handle the request due to reasons such as temporary overloading, maintenance of the server, or rate limiting of the client.

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 9.7). If no Retry-After is given, the client SHOULD handle the response as it would for a 500 response.

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

8.5.5. 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 implementors: some deployed proxies are known to return 400 or 500 when DNS lookups time out.

8.5.6. 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 2.6 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.

9. Header Field Definitions

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

9.1. 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.

9.2. Expect

The "Expect" header field is used to indicate that particular server behaviors are required by the client.

Expect = 1#expectation

expectation = "100-continue" / expectation-extension
expectation-extension = token ["=" (token / quoted-string)
*expect-params]
expect-params = ";" token ["=" (token / quoted-string)]

A server that does not understand or is unable to comply with any of

the expectation values in the Expect field of a request MUST respond with appropriate error status code. The server MUST respond with a 417 (Expectation Failed) status code if any of the expectations cannot be met or, if there are other problems with the request, some other 4xx status code.

This header field is defined with extensible syntax to allow for future extensions. If a server receives a request containing an Expect field that includes an expectation-extension that it does not support, it MUST respond with a 417 (Expectation Failed) status code.

Comparison of expectation values is case-insensitive for unquoted tokens (including the 100-continue token), and is case-sensitive for quoted-string expectation-extensions.

The Expect mechanism is hop-by-hop: that is, an HTTP/1.1 proxy MUST return a 417 (Expectation Failed) status code if it receives a request with an expectation that it cannot meet. 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.

See Section 7.2.3 of [Part1] for the use of the 100 (Continue) status code.

9.3. 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.

9.4. Location

The "Location" header field is used to identify a newly created resource, or to redirect the recipient to a different location for completion of the request.

For 201 (Created) responses, the Location is the URI of the new resource which was created by the request. For 3xx 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).

Location = URI-reference

Examples are:

Location: <http://www.example.org/pub/WWW/People.html#tim>

Location: /index.html

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: This specification does not define precedence rules for the case where the original URI, as navigated to by the user agent, and the Location header field value both contain fragment identifiers. Thus be aware that including fragment identifiers might inconvenience anyone relying on the semantics of the

original URI's fragment identifier.

Note: The Content-Location header field (Section 6.7 of [Part3]) 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.

9.5. Max-Forwards

The "Max-Forwards" header field provides a mechanism with the TRACE (Section 7.8) and OPTIONS (Section 7.2) 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 in 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.

9.6. Referer

The "Referer" [sic] header field allows the client to specify the URI of the resource from which the effective request 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 effective request 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.

9.7. 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 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.

9.8. 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 6.3 of [Part1]) 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 9.9 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 implementors are encouraged to make this field a configurable option.

9.9. 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 6.3 of [Part1]) 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	Reference
CONNECT	no	Section 7.9
DELETE	no	Section 7.7
GET	yes	Section 7.3
HEAD	yes	Section 7.4
OPTIONS	yes	Section 7.2
POST	no	Section 7.5
PUT	no	Section 7.6
TRACE	yes	Section 7.8

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 8.1.1
101	Switching Protocols	Section 8.1.2
200	OK	Section 8.2.1
201	Created	Section 8.2.2
202	Accepted	Section 8.2.3
203	Non-Authoritative Information	Section 8.2.4
204	No Content	Section 8.2.5
205	Reset Content	Section 8.2.6
300	Multiple Choices	Section 8.3.1
301	Moved Permanently	Section 8.3.2
302	Found	Section 8.3.3
303	See Other	Section 8.3.4
305	Use Proxy	Section 8.3.6
306	(Unused)	Section 8.3.7
307	Temporary Redirect	Section 8.3.8
400	Bad Request	Section 8.4.1
402	Payment Required	Section 8.4.3
403	Forbidden	Section 8.4.4
404	Not Found	Section 8.4.5
405	Method Not Allowed	Section 8.4.6
406	Not Acceptable	Section 8.4.7
407	Proxy Authentication Required	Section 8.4.8
408	Request Timeout	Section 8.4.9
409	Conflict	Section 8.4.10
410	Gone	Section 8.4.11
411	Length Required	Section 8.4.12
413	Request Representation Too Large	Section 8.4.14
414	URI Too Long	Section 8.4.15
415	Unsupported Media Type	Section 8.4.16
417	Expectation Failed	Section 8.4.18
426	Upgrade Required	Section 8.4.19
500	Internal Server Error	Section 8.5.1
501	Not Implemented	Section 8.5.2
502	Bad Gateway	Section 8.5.3
503	Service Unavailable	Section 8.5.4
504	Gateway Timeout	Section 8.5.5
505	HTTP Version Not Supported	Section 8.5.6

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
Allow	http	standard	Section 9.1
Expect	http	standard	Section 9.2
From	http	standard	Section 9.3
Location	http	standard	Section 9.4
Max-Forwards	http	standard	Section 9.5
Referer	http	standard	Section 9.6
Retry-After	http	standard	Section 9.7
Server	http	standard	Section 9.8
User-Agent	http	standard	Section 9.9

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

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. Implementors 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 9.9) or Server (Section 9.8) header fields can sometimes be used to determine that a specific client or server have 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 may contain enough entropy to be used, possibly in conjunction with other material, to uniquely identify the user.

Some request methods, like TRACE (Section 7.8), 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 Headers and Spoofing

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.

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 **CONNECT**ing to port 25 could relay spam via SMTP, for example. As such, proxies **SHOULD** restrict **CONNECT** access to a small number of known ports.

12. Acknowledgments

13. References

13.1. Normative References

- [Part1] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing", draft-ietf-httpbis-p1-messaging-15 (work in progress), July 2011.
- [Part3] 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 3: Message Payload and Content Negotiation", draft-ietf-httpbis-p3-payload-15 (work in progress), July 2011.
- [Part4] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 4: Conditional Requests", draft-ietf-httpbis-p4-conditional-15 (work in progress), July 2011.
- [Part5] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 5: Range Requests and

Partial Responses", draft-ietf-httpbis-p5-range-15 (work in progress), July 2011.

- [Part6] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1, part 6: Caching", draft-ietf-httpbis-p6-cache-15 (work in progress), July 2011.
- [Part7] 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-15 (work in progress), July 2011.
- [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.

13.2. Informative References

- [RFC1945] Berners-Lee, T., Fielding, R., and H. Nielsen, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.
- [RFC2068] Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 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.
- [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.

[RFC5322] Resnick, P., "Internet Message Format", RFC 5322, October 2008.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, March 2010.

Appendix A. Changes from RFC 2616

This document takes over the Status Code Registry, previously defined in Section 7.1 of [RFC2817]. (Section 4.2)

Clarify definition of POST. (Section 7.5)

Remove requirement to handle all Content-* header fields; ban use of Content-Range with PUT. (Section 7.6)

Take over definition of CONNECT method from [RFC2817]. (Section 7.9)

Broadened the definition of 203 (Non-Authoritative Information) to include cases of payload transformations as well. (Section 8.2.4)

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. (Sections 8.3.2, 8.3.3 and 8.3.8)

Deprecate 305 Use Proxy status code, because user agents did not implement it. It used to indicate that the target resource must 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 8.3.6)

Define status 426 (Upgrade Required) (this was incorporated from [RFC2817]). (Section 8.4.19)

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 9.1)

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 9.4)

Restrict Max-Forwards header field to OPTIONS and TRACE (previously, extension methods could have used it as well). (Section 9.5)

Allow Referer field value of "about:blank" as alternative to not specifying it. (Section 9.6)

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 9.9 of [Part1]. (Section 9.8)

Appendix B. Collected ABNF

Allow = [("," / Method) *(OWS "," [OWS Method])]

Expect = *("," OWS) expectation *(OWS "," [OWS expectation])

From = mailbox

HTTP-date = <HTTP-date, defined in [Part1], Section 6.1>

Location = URI-reference

Max-Forwards = 1*DIGIT

Method = token

OWS = <OWS, defined in [Part1], Section 1.2.2>

RWS = <RWS, defined in [Part1], Section 1.2.2>

Reason-Phrase = *(WSP / VCHAR / obs-text)

Referer = absolute-URI / partial-URI

Retry-After = HTTP-date / delta-seconds

Server = product *(RWS (product / comment))

Status-Code = 3DIGIT

URI-reference = <URI-reference, defined in [Part1], Section 2.7>

User-Agent = product *(RWS (product / comment))

absolute-URI = <absolute-URI, defined in [Part1], Section 2.7>

comment = <comment, defined in [Part1], Section 3.2>

delta-seconds = 1*DIGIT

expect-params = ";" token ["=" (token / quoted-string)]

expectation = "100-continue" / expectation-extension

expectation-extension = token ["=" (token / quoted-string)
*expect-params]

mailbox = <mailbox, defined in [RFC5322], Section 3.4>

obs-text = <obs-text, defined in [Part1], Section 1.2.2>

partial-URI = <partial-URI, defined in [Part1], Section 2.7>

product = <product, defined in [Part1], Section 6.3>

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

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

ABNF diagnostics:

```
; Allow defined but not used
; Expect defined but not used
; From defined but not used
; Location defined but not used
; Max-Forwards defined but not used
; Reason-Phrase defined but not used
; Referer defined but not used
; Retry-After defined but not used
; Server defined but not used
; Status-Code defined but not used
; User-Agent defined but not used
```

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

C.1. Since RFC 2616

Extracted relevant partitions from [RFC2616].

C.2. Since draft-ietf-httpbis-p2-semantics-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]

C.3. 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.

C.4. 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"
- 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 headers 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).

C.5. Since draft-ietf-httpbis-p2-semantic-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).

C.6. Since draft-ietf-httpbis-p2-semantic-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.

C.7. 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.

C.8. 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"

C.9. 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 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?"

C.10. 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)

C.11. 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 payload handling"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/196>>: "Term for the requested resource's URI"

C.12. 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"

C.13. 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)

C.14. 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 may 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 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 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 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"

C.15. 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"

C.16. Since draft-ietf-httpbis-p2-semantics-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"

Index

1

- 100 Continue (status code) 23
- 101 Switching Protocols (status code) 23

2

- 200 OK (status code) 24
- 201 Created (status code) 24
- 202 Accepted (status code) 25
- 203 Non-Authoritative Information (status code) 25
- 204 No Content (status code) 25
- 205 Reset Content (status code) 26
- 206 Partial Content (status code) 26

3

- 300 Multiple Choices (status code) 27
- 301 Moved Permanently (status code) 27
- 302 Found (status code) 28
- 303 See Other (status code) 28
- 304 Not Modified (status code) 29
- 305 Use Proxy (status code) 29

	306 (Unused) (status code)	29
	307 Temporary Redirect (status code)	29
4		
	400 Bad Request (status code)	30
	401 Unauthorized (status code)	30
	402 Payment Required (status code)	30
	403 Forbidden (status code)	30
	404 Not Found (status code)	31
	405 Method Not Allowed (status code)	31
	406 Not Acceptable (status code)	31
	407 Proxy Authentication Required (status code)	32
	408 Request Timeout (status code)	32
	409 Conflict (status code)	32
	410 Gone (status code)	32
	411 Length Required (status code)	33
	412 Precondition Failed (status code)	33
	413 Request Representation Too Large (status code)	33
	414 URI Too Long (status code)	33
	415 Unsupported Media Type (status code)	34
	416 Requested Range Not Satisfiable (status code)	34
	417 Expectation Failed (status code)	34
	426 Upgrade Required (status code)	34
5		
	500 Internal Server Error (status code)	35
	501 Not Implemented (status code)	35
	502 Bad Gateway (status code)	35
	503 Service Unavailable (status code)	35
	504 Gateway Timeout (status code)	35
	505 HTTP Version Not Supported (status code)	36
A		
	Allow header field	36
C		
	CONNECT method	21
D		
	DELETE method	20
E		
	Expect header field	36
F		
	From header field	37
G		

GET method 16
Grammar
 Allow 36
 delta-seconds 40
 Expect 36
 expect-params 36
 expectation 36
 expectation-extension 36
 extension-code 10
 From 37
 Location 38
 Max-Forwards 39
 Method 7
 Reason-Phrase 10
 Referer 40
 Retry-After 40
 Server 40
 Status-Code 10
 User-Agent 41

H

HEAD method 17
Header Fields
 Allow 36
 Expect 36
 From 37
 Location 38
 Max-Forwards 39
 Referer 39
 Retry-After 40
 Server 40
 User-Agent 41

I

Idempotent Methods 15

L

Location header field 38

M

Max-Forwards header field 39
Methods
 CONNECT 21
 DELETE 20
 GET 16
 HEAD 17
 OPTIONS 15
 POST 17

PUT	18
TRACE	21
O	
OPTIONS method	15
P	
POST method	17
PUT method	18
R	
Referer header field	39
Retry-After header field	40
S	
Safe Methods	14
Server header field	40
Status Codes	
100 Continue	23
101 Switching Protocols	23
200 OK	24
201 Created	24
202 Accepted	25
203 Non-Authoritative Information	25
204 No Content	25
205 Reset Content	26
206 Partial Content	26
300 Multiple Choices	27
301 Moved Permanently	27
302 Found	28
303 See Other	28
304 Not Modified	29
305 Use Proxy	29
306 (Unused)	29
307 Temporary Redirect	29
400 Bad Request	30
401 Unauthorized	30
402 Payment Required	30
403 Forbidden	30
404 Not Found	31
405 Method Not Allowed	31
406 Not Acceptable	31
407 Proxy Authentication Required	32
408 Request Timeout	32
409 Conflict	32
410 Gone	32
411 Length Required	33
412 Precondition Failed	33

413 Request Representation Too Large	33
414 URI Too Long	33
415 Unsupported Media Type	34
416 Requested Range Not Satisfiable	34
417 Expectation Failed	34
426 Upgrade Required	34
500 Internal Server Error	35
501 Not Implemented	35
502 Bad Gateway	35
503 Service Unavailable	35
504 Gateway Timeout	35
505 HTTP Version Not Supported	36

T

TRACE method 21

U

User-Agent header field 41

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/>

Jim Gettys
Alcatel-Lucent Bell Labs
21 Oak Knoll Road
Carlisle, MA 01741
USA

EMail: jg@freedesktop.org
URI: <http://gettys.wordpress.com/>

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

EMail: JeffMogul@acm.org

Henrik Frystyk Nielsen
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052
USA

EMail: henrikn@microsoft.com

Larry Masinter
Adobe Systems Incorporated
345 Park Ave
San Jose, CA 95110
USA

EMail: LMM@acm.org
URI: <http://larry.masinter.net/>

Paul J. Leach
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052

EMail: paulle@microsoft.com

Tim Berners-Lee
World Wide Web Consortium
MIT Computer Science and Artificial Intelligence Laboratory
The Stata Center, Building 32
32 Vassar Street
Cambridge, MA 02139
USA

EMail: timbl@w3.org
URI: <http://www.w3.org/People/Berners-Lee/>

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

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

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

Phone: +49 251 2807760
Fax: +49 251 2807761
EMail: julian.reschke@greenbytes.de
URI: <http://greenbytes.de/tech/webdav/>

HTTPbis Working Group
Internet-Draft
Obsoletes: 2616 (if approved)
Intended status: Standards Track
Expires: January 12, 2012

R. Fielding, Ed.
Adobe
J. Gettys
Alcatel-Lucent
J. Mogul
HP
H. Frystyk
Microsoft
L. Masinter
Adobe
P. Leach
Microsoft
T. Berners-Lee
W3C/MIT
Y. Lafon, Ed.
W3C
J. Reschke, Ed.
greenbytes
July 11, 2011

HTTP/1.1, part 3: Message Payload and Content Negotiation
draft-ietf-httpbis-p3-payload-15

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World Wide Web global information initiative since 1990. This document is Part 3 of the seven-part specification that defines the protocol referred to as "HTTP/1.1" and, taken together, obsoletes RFC 2616. Part 3 defines HTTP message content, metadata, and content negotiation.

Editorial Note (To be removed by RFC Editor)

Discussion of this draft should take place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org), 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.16.

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 12, 2012.

Copyright Notice

Copyright (c) 2011 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	5
1.1. Terminology	5
1.2. Requirements	5
1.3. Syntax Notation	6

1.3.1. Core Rules	6
1.3.2. ABNF Rules defined in other Parts of the Specification	6
2. Protocol Parameters	6
2.1. Character Encodings (charset)	6
2.2. Content Codings	7
2.2.1. Content Coding Registry	8
2.3. Media Types	8
2.3.1. Canonicalization and Text Defaults	9
2.3.2. Multipart Types	9
2.4. Language Tags	10
3. Payload	10
3.1. Payload Header Fields	11
3.2. Payload Body	11
4. Representation	11
4.1. Representation Header Fields	12
4.2. Representation Data	12
5. Content Negotiation	13
5.1. Server-driven Negotiation	14
5.2. Agent-driven Negotiation	15
6. Header Field Definitions	16
6.1. Accept	16
6.2. Accept-Charset	18
6.3. Accept-Encoding	19
6.4. Accept-Language	20
6.5. Content-Encoding	21
6.6. Content-Language	22
6.7. Content-Location	23
6.8. Content-Type	24
7. IANA Considerations	24
7.1. Header Field Registration	24
7.2. Content Coding Registry	25
8. Security Considerations	25
8.1. Privacy Issues Connected to Accept Header Fields	26
9. Acknowledgments	26
10. References	26
10.1. Normative References	26
10.2. Informative References	28
Appendix A. Differences between HTTP and MIME	29
A.1. MIME-Version	30
A.2. Conversion to Canonical Form	30
A.3. Conversion of Date Formats	31
A.4. Introduction of Content-Encoding	31
A.5. No Content-Transfer-Encoding	31
A.6. Introduction of Transfer-Encoding	31
A.7. MHTML and Line Length Limitations	31
Appendix B. Additional Features	32
Appendix C. Changes from RFC 2616	32

Appendix D. Collected ABNF	33
Appendix E. Change Log (to be removed by RFC Editor before publication)	34
E.1. Since RFC 2616	34
E.2. Since draft-ietf-httpbis-p3-payload-00	34
E.3. Since draft-ietf-httpbis-p3-payload-01	35
E.4. Since draft-ietf-httpbis-p3-payload-02	35
E.5. Since draft-ietf-httpbis-p3-payload-03	35
E.6. Since draft-ietf-httpbis-p3-payload-04	36
E.7. Since draft-ietf-httpbis-p3-payload-05	36
E.8. Since draft-ietf-httpbis-p3-payload-06	37
E.9. Since draft-ietf-httpbis-p3-payload-07	37
E.10. Since draft-ietf-httpbis-p3-payload-08	37
E.11. Since draft-ietf-httpbis-p3-payload-09	38
E.12. Since draft-ietf-httpbis-p3-payload-10	38
E.13. Since draft-ietf-httpbis-p3-payload-11	39
E.14. Since draft-ietf-httpbis-p3-payload-12	39
E.15. Since draft-ietf-httpbis-p3-payload-13	39
E.16. Since draft-ietf-httpbis-p3-payload-14	40
Index	40

1. Introduction

This document defines HTTP/1.1 message payloads (a.k.a., content), the associated metadata header fields that define how the payload is intended to be interpreted by a recipient, the request header fields that might influence content selection, and the various selection algorithms that are collectively referred to as HTTP content negotiation.

This document is currently disorganized in order to minimize the changes between drafts and enable reviewers to see the smaller errata changes. A future draft will reorganize the sections to better reflect the content. In particular, the sections on entities will be renamed payload and moved to the first half of the document, while the sections on content negotiation and associated request header fields will be moved to the second half. The current mess reflects how widely dispersed these topics and associated requirements had become in [RFC2616].

1.1. Terminology

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

content negotiation

The mechanism for selecting the appropriate representation when servicing a request. The representation in any response can be negotiated (including error responses).

1.2. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

An implementation is not compliant if it fails to satisfy one or more of the "MUST" or "REQUIRED" level requirements for the protocols it implements. An implementation that satisfies all the "MUST" or "REQUIRED" level and all the "SHOULD" level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the "MUST" level requirements but not all the "SHOULD" level requirements for its protocols is said to be "conditionally compliant".

1.3. Syntax Notation

This specification uses the ABNF syntax defined in Section 1.2 of [Part1] (which extends the syntax defined in [RFC5234] with a list rule). Appendix D shows the collected ABNF, with the list rule expanded.

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

1.3.1. Core Rules

The core rules below are defined in Section 1.2.2 of [Part1]:

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

1.3.2. ABNF Rules defined in other Parts of the Specification

The ABNF rules below are defined in other parts:

absolute-URI	= <absolute-URI, defined in [Part1], Section 2.7>
partial-URI	= <partial-URI, defined in [Part1], Section 2.7>
qvalue	= <qvalue, defined in [Part1], Section 6.4>

2. Protocol Parameters

2.1. 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.

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

2.2. 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 6.3) and Content-Encoding (Section 6.5) header fields. Although the value describes the content-coding, what is more important is that it indicates what decoding mechanism will be required to remove the encoding.

compress

See Section 6.2.2.1 of [Part1].

deflate

See Section 6.2.2.2 of [Part1].

gzip

See Section 6.2.2.3 of [Part1].

identity

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

2.2.1. Content Coding Registry

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

Registrations MUST include the following fields:

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

Names of content codings MUST NOT overlap with names of transfer codings (Section 6.2 of [Part1]), unless the encoding transformation is identical (as it is the case for the compression codings defined in Section 6.2.2 of [Part1]).

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

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

2.3. Media Types

HTTP uses Internet Media Types [RFC2046] in the Content-Type (Section 6.8) and Accept (Section 6.1) 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.

2.3.1. 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.

2.3.2. 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].

2.4. Language Tags

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

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

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

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

Example tags include:

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

See [RFC5646] for further information.

3. 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 status code).

3.1. 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 9.2 of [Part1]
Content-Range	Section 5.2 of [Part5]

3.2. 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.

4. 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.

4.1. 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 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 6.5
Content-Language	Section 6.6
Content-Location	Section 6.7
Content-Type	Section 6.8
Expires	Section 3.3 of [Part6]
Last-Modified	Section 2.1 of [Part4]

4.2. 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.

5. 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".

5.1. 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.

HTTP/1.1 includes the following header fields for enabling server-driven negotiation through description of user agent capabilities and user preferences: Accept (Section 6.1), Accept-Charset (Section 6.2), Accept-Encoding (Section 6.3), Accept-Language (Section 6.4), and User-Agent (Section 9.9 of [Part2]). However, an origin server is not limited to these dimensions and MAY vary the response based on any aspect of the request, including 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 3.5 of [Part6]) can be used to express the parameters the server uses to select a representation that is subject to server-driven negotiation.

5.2. 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.

6. Header Field Definitions

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

6.1. 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 6.4 of [Part1]). The default value is q=1.

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

The example

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

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

If no Accept header field is present, then it is assumed that the client accepts all media types. If an Accept header field is present, and if the server cannot send a response which is acceptable according to the combined Accept field value, then the server SHOULD send a 406 (Not Acceptable) response.

A more elaborate example is

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

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

6.2. 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 2.1. Each charset MAY be given an associated quality value which represents the user's preference for that charset. The default value is q=1. An example is

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

The special value "*", if present in the Accept-Charset field, matches every character 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.

If no Accept-Charset header field is present, the default is that any character encoding is acceptable. If an Accept-Charset header field is present, and if the server cannot send a response which is acceptable according to the Accept-Charset header field, then the server SHOULD send an error response with the 406 (Not Acceptable) status code, though the sending of an unacceptable response is also allowed.

6.3. Accept-Encoding

The "Accept-Encoding" header field can be used by user agents to indicate what response content-codings (Section 2.2) are acceptable in the response.

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

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

Examples of its use are:

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

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

1. If the content-coding is one of the content-codings listed in the Accept-Encoding field, then it is acceptable, unless it is accompanied by a qvalue of 0. (As defined in Section 6.4 of [Part1], a qvalue of 0 means "not acceptable".)
2. The special "*" symbol in an Accept-Encoding field matches any available content-coding not explicitly listed in the header field.
3. If multiple content-codings are acceptable, then the acceptable content-coding with the highest non-zero qvalue is preferred.
4. The "identity" content-coding is always acceptable, unless specifically refused because the Accept-Encoding field includes "identity;q=0", or because the field includes "*;q=0" and does not explicitly include the "identity" content-coding. If the Accept-Encoding field-value is empty, then only the "identity" encoding is acceptable.

If an Accept-Encoding field is present in a request, and if the server cannot send a response which is acceptable according to the Accept-Encoding header field, then the server SHOULD send an error response with the 406 (Not Acceptable) status code.

If no Accept-Encoding field is present in a request, the server MAY assume that the client will accept any content coding. In this case, if "identity" is one of the available content-codings, then the server SHOULD use the "identity" content-coding, unless it has additional information that a different content-coding is meaningful to the client.

Note: If the request does not include an Accept-Encoding field, and if the "identity" content-coding is unavailable, then content-codings commonly understood by HTTP/1.0 clients (i.e., "gzip" and "compress") are preferred; some older clients improperly display messages sent with other content-codings. The server might also make this decision based on information about the particular user-agent or client.

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

6.4. 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 2.4.

```
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 8.1.

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

Note: When making the choice of linguistic preference available to the user, we remind implementors of the fact that users are not familiar with the details of language matching as described above, and 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.

6.5. Content-Encoding

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

Content-Encoding = 1#content-coding

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

Content-Encoding: gzip

The content-coding is a characteristic of the 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 content-coding of a representation is not "identity", then the representation metadata **MUST** include a Content-Encoding header field (Section 6.5) that lists the non-identity content-coding(s) used.

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).

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.

6.6. 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 2.4. 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.

6.7. 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 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 4.3 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 the 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 representation selected 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 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.

6.8. 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 2.3. An example of the field is

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

Further discussion of Content-Type is provided in Section 4.2.

7. IANA Considerations

7.1. Header Field Registration

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

Header Field Name	Protocol	Status	Reference
Accept	http	standard	Section 6.1
Accept-Charset	http	standard	Section 6.2
Accept-Encoding	http	standard	Section 6.3
Accept-Language	http	standard	Section 6.4
Content-Encoding	http	standard	Section 6.5
Content-Language	http	standard	Section 6.6
Content-Location	http	standard	Section 6.7
Content-Type	http	standard	Section 6.8
MIME-Version	http	standard	Appendix A.1

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

7.2. Content Coding Registry

The registration procedure for HTTP Content Codings is now defined by Section 2.2.1 of this document.

The HTTP Content Codings Registry located at <http://www.iana.org/assignments/http-parameters> shall be updated with the registration below:

Name	Description	Reference
compress	UNIX "compress" program method	Section 6.2.2.1 of [Part1]
deflate	"deflate" compression mechanism ([RFC1951]) used inside the "zlib" data format ([RFC1950])	Section 6.2.2.2 of [Part1]
gzip	Same as GNU zip [RFC1952]	Section 6.2.2.3 of [Part1]
identity	No transformation	Section 2.2

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. Privacy Issues Connected to Accept Header Fields

Accept headers 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 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 configurability SHOULD warn users about the loss of privacy which can be involved.

9. Acknowledgments

10. References

10.1. Normative References

- [Part1] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing", draft-ietf-httpbis-pl-messaging-15 (work in progress), July 2011.
- [Part2] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H.,

Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 2: Message Semantics", draft-ietf-httpbis-p2-semantics-15 (work in progress), July 2011.

[Part4] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 4: Conditional Requests", draft-ietf-httpbis-p4-conditional-15 (work in progress), July 2011.

[Part5] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 5: Range Requests and Partial Responses", draft-ietf-httpbis-p5-range-15 (work in progress), July 2011.

[Part6] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1, part 6: Caching", draft-ietf-httpbis-p6-cache-15 (work in progress), July 2011.

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

RFC 1950 is an Informational RFC, thus it might be less stable than this specification. On the other hand, this downward reference was present since the publication of RFC 2068 in 1997 ([RFC2068]), therefore it is unlikely to cause problems in practice. See also [BCP97].

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

RFC 1951 is an Informational RFC, thus it might be less stable than this specification. On the other hand, this downward reference was present since the publication of RFC 2068 in 1997 ([RFC2068]), therefore it is unlikely to cause problems in practice. See also [BCP97].

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

RFC 1952 is an Informational RFC, thus it might be less stable than this specification. On the other hand, this downward reference was present since the publication of

RFC 2068 in 1997 ([RFC2068]), therefore it is unlikely to cause problems in practice. See also [BCP97].

- [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.
- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4647] Phillips, A., Ed. and M. Davis, Ed., "Matching of Language Tags", BCP 47, RFC 4647, September 2006.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, September 2009.

10.2. Informative References

- [BCP97] Klensin, J. and S. Hartman, "Handling Normative References to Standards-Track Documents", BCP 97, RFC 4897, June 2007.
- [RFC1945] Berners-Lee, T., Fielding, R., and H. Nielsen, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.
- [RFC2049] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples", RFC 2049, November 1996.
- [RFC2068] Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997.
- [RFC2076] Palme, J., "Common Internet Message Headers", RFC 2076, February 1997.
- [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.
- [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.
- [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.

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 compliance with the MIME protocol (as defined in [RFC2045]). Proxies/gateways are responsible for ensuring full compliance (where possible) when exporting HTTP messages to strict MIME environments.

MIME-Version = 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 2.3.1 of this document describes the forms allowed for subtypes of the "text" media type when transmitted over HTTP. [RFC2046] requires that content with a type of "text" represent line breaks as CRLF and forbids the use of CR or LF outside of line break sequences. HTTP allows CRLF, bare CR, and bare LF to indicate a line break within text content when a message is transmitted over HTTP.

Where it is possible, a proxy or gateway from HTTP to a strict MIME environment SHOULD translate all line breaks within the text media types described in Section 2.3.1 of this document to the RFC 2049 canonical form of CRLF. Note, however, that this might be complicated by the presence of a Content-Encoding and by the fact that HTTP allows the use of some character 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 6.1 of [Part1]) to simplify the process of date comparison. Proxies and gateways from other protocols SHOULD ensure that any Date header field present in a message conforms to one of the HTTP/1.1 formats and rewrite the date if necessary.

A.4. Introduction of Content-Encoding

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. Introduction of Transfer-Encoding

HTTP/1.1 introduces the Transfer-Encoding header field (Section 9.7 of [Part1]). Proxies/gateways MUST remove any transfer-coding prior to forwarding a message via a MIME-compliant protocol.

A.7. MHTML and Line Length Limitations

HTTP implementations which share code with MHTML [RFC2557] implementations need to be aware of MIME line length limitations. Since HTTP does not have this limitation, HTTP does not fold long lines. MHTML messages being transported by HTTP follow all conventions of MHTML, including line length limitations and folding,

canonicalization, etc., since HTTP transports all message-bodies as payload (see Section 2.3.2) and does not interpret the content or any MIME header lines that might be contained therein.

Appendix B. Additional Features

[RFC1945] and [RFC2068] document protocol elements used by some existing HTTP implementations, but not consistently and correctly across most HTTP/1.1 applications. Implementors are advised to be aware of these features, but cannot rely upon their presence in, or interoperability with, other HTTP/1.1 applications. Some of these describe proposed experimental features, and some describe features that experimental deployment found lacking that are now addressed in the base HTTP/1.1 specification.

A number of other 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

Clarify contexts that charset is used in. (Section 2.1)

Remove the default character encoding for text media types; the default now is whatever the media type definition says. (Section 2.3.1)

Change ABNF productions for header fields to only define the field value. (Section 6)

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 6)

Remove ISO-8859-1 special-casing in Accept-Charset. (Section 6.2)

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 6.7)

Remove discussion of Content-Disposition header field, it is now defined by [RFC6266]. (Appendix B)

Remove reference to non-existent identity transfer-coding value tokens. (Appendix A.5)

Appendix D. 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 ]
    ] )

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

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

OWS = <OWS, defined in [Part1], Section 1.2.2>

absolute-URI = <absolute-URI, defined in [Part1], Section 2.7>
accept-ext = OWS ";" OWS token [ "=" word ]
accept-params = OWS ";" OWS "q=" qvalue *accept-ext
attribute = token

charset = token
codings = ( content-coding / "*" )
content-coding = token

language-range = <language-range, defined in [RFC4647], Section 2.1>
language-tag = <Language-Tag, defined in [RFC5646], Section 2.1>

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

parameter = attribute "=" value
partial-URI = <partial-URI, defined in [Part1], Section 2.7>

qvalue = <qvalue, defined in [Part1], Section 6.4>

subtype = token

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

type = token

value = word

word = <word, defined in [Part1], Section 1.2.2>

ABNF diagnostics:

```
; Accept defined but not used
; Accept-Charset defined but not used
; Accept-Encoding defined but not used
; Accept-Language defined but not used
; Content-Encoding defined but not used
; Content-Language defined but not used
; Content-Location defined but not used
; Content-Type defined but not used
; MIME-Version defined but not used
```

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

E.1. Since RFC 2616

Extracted relevant partitions from [RFC2616].

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

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/8>>: "Media Type Registrations" (<<http://purl.org/NET/http-errata#media-reg>>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/14>>: "Clarification regarding quoting of charset values" (<<http://purl.org/NET/http-errata#charactersets>>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/16>>: "Remove 'identity' token references" (<<http://purl.org/NET/http-errata#identity>>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/25>>: "Accept-Encoding BNF"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/35>>: "Normative and Informative references"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/46>>: "RFC1700 references"

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/55>>: "Updating to RFC4288"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/65>>: "Informative references"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/66>>: "ISO-8859-1 Reference"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/68>>: "Encoding References Normative"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/86>>: "Normative up-to-date references"

E.3. Since draft-ietf-httpbis-p3-payload-01

Ongoing work on ABNF conversion

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

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

E.4. Since draft-ietf-httpbis-p3-payload-02

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/67>>: "Quoting Charsets"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/105>>: "Classification for Allow header"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/115>>: "missing default for qvalue in description of Accept-Encoding"

Ongoing work on IANA Message Header Field Registration

(<<http://tools.ietf.org/wg/httpbis/trac/ticket/40>>):

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

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

Closed issues:

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

- o <http://tools.ietf.org/wg/httpbis/trac/ticket/113>: "language tag matching (Accept-Language) vs RFC4647"
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/121>: "RFC 1806 has been replaced by RFC2183"

Other changes:

- o <http://tools.ietf.org/wg/httpbis/trac/ticket/68>: "Encoding References Normative" -- rephrase the annotation and reference [BCP97].

E.6. Since draft-ietf-httpbis-p3-payload-04

Closed issues:

- o <http://tools.ietf.org/wg/httpbis/trac/ticket/132>: "RFC 2822 is updated by RFC 5322"

Ongoing work on ABNF conversion

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

- o Use "/" instead of "|" for alternatives.
- o Introduce new ABNF rules for "bad" whitespace ("BWS"), optional whitespace ("OWS") and required whitespace ("RWS").
- o Rewrite ABNFs to spell out whitespace rules, factor out header field value format definitions.

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

Closed issues:

- o <http://tools.ietf.org/wg/httpbis/trac/ticket/118>: "Join "Differences Between HTTP Entities and RFC 2045 Entities"?"

Final work on ABNF conversion

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

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

Other changes:

- o Move definition of quality values into Part 1.

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

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/80>>: "Content-Location isn't special"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/155>>: "Content Sniffing"

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

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/13>>: "Updated reference for language tags"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/110>>: "Clarify rules for determining what entities a response carries"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/154>>: "Content-Location base-setting problems"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/155>>: "Content Sniffing"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/188>>: "pick IANA policy (RFC5226) for Transfer Coding / Content Coding"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/189>>: "move definitions of gzip/deflate/compress to part 1"

Partly resolved issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/148>>: "update IANA requirements wrt Transfer-Coding values" (add the IANA Considerations subsection)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/149>>: "update IANA requirements wrt Content-Coding values" (add the IANA Considerations subsection)

E.10. 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?"

E.11. 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 structure"

Partly resolved issues:

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

E.12. 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"

E.13. Since draft-ietf-httpbis-p3-payload-11

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/123>>: "Factor out Content-Disposition"

E.14. Since draft-ietf-httpbis-p3-payload-12

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/224>>: "Header 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"

E.15. 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"

E.16. Since draft-ietf-httpbis-p3-payload-14

None.

Index

A

- Accept header field 16
- Accept-Charset header field 18
- Accept-Encoding header field 19
- Accept-Language header field 20

C

- Coding Format
 - compress 7
 - deflate 7
 - gzip 7
 - identity 7
- compress (Coding Format) 7
- content negotiation 5
- Content-Encoding header field 21
- Content-Language header field 22
- Content-Location header field 23
- Content-Type header field 24

D

- deflate (Coding Format) 7

G

- Grammar
 - Accept 16
 - Accept-Charset 18
 - Accept-Encoding 19
 - accept-ext 16
 - Accept-Language 20
 - accept-params 16
 - attribute 8
 - charset 6
 - codings 19
 - content-coding 7
 - Content-Encoding 21
 - Content-Language 22
 - Content-Location 23
 - Content-Type 24
 - language-range 20
 - language-tag 10
 - media-range 16
 - media-type 8

MIME-Version 30
 parameter 8
 subtype 8
 type 8
 value 8
gzip (Coding Format) 7

H

Header Fields
 Accept 16
 Accept-Charset 18
 Accept-Encoding 19
 Accept-Language 20
 Content-Encoding 21
 Content-Language 22
 Content-Location 23
 Content-Type 24
 MIME-Version 30

I

identity (Coding Format) 7

M

MIME-Version header field 30

P

payload 10

R

representation 11

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/>

Jim Gettys
Alcatel-Lucent Bell Labs
21 Oak Knoll Road
Carlisle, MA 01741
USA

EMail: jg@freedesktop.org
URI: <http://gettys.wordpress.com/>

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

EMail: JeffMogul@acm.org

Henrik Frystyk Nielsen
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052
USA

EMail: henrikn@microsoft.com

Larry Masinter
Adobe Systems Incorporated
345 Park Ave
San Jose, CA 95110
USA

EMail: LMM@acm.org
URI: <http://larry.masinter.net/>

Paul J. Leach
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052

EMail: paulle@microsoft.com

Tim Berners-Lee
World Wide Web Consortium
MIT Computer Science and Artificial Intelligence Laboratory
The Stata Center, Building 32
32 Vassar Street
Cambridge, MA 02139
USA

EMail: timbl@w3.org
URI: <http://www.w3.org/People/Berners-Lee/>

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

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

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

Phone: +49 251 2807760
Fax: +49 251 2807761
EMail: julian.reschke@greenbytes.de
URI: <http://greenbytes.de/tech/webdav/>

HTTPbis Working Group
Internet-Draft
Obsoletes: 2616 (if approved)
Intended status: Standards Track
Expires: January 12, 2012

R. Fielding, Ed.
Adobe
J. Gettys
Alcatel-Lucent
J. Mogul
HP
H. Frystyk
Microsoft
L. Masinter
Adobe
P. Leach
Microsoft
T. Berners-Lee
W3C/MIT
Y. Lafon, Ed.
W3C
J. Reschke, Ed.
greenbytes
July 11, 2011

HTTP/1.1, part 4: Conditional Requests
draft-ietf-httpbis-p4-conditional-15

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World Wide Web global information initiative since 1990. This document is Part 4 of the seven-part specification that defines the protocol referred to as "HTTP/1.1" and, taken together, obsoletes RFC 2616. Part 4 defines request header fields for indicating conditional requests and the rules for constructing responses to those requests.

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://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 C.16.

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 12, 2012.

Copyright Notice

Copyright (c) 2011 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	5
1.1. Requirements	5

1.2. Syntax Notation	6
2. Resource State Metadata (Validators)	6
2.1. Last-Modified	6
2.1.1. Generation	6
2.1.2. Comparison	7
2.2. ETag	8
2.2.1. Generation	9
2.2.2. Weak versus Strong	9
2.2.3. Comparison	11
2.2.4. Rules for When to Use Entity-tags and Last-Modified Dates	11
2.2.5. Example: Entity-tags varying on Content-Negotiated Resources	13
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	18
3.5. If-Range	18
4. Status Code Definitions	18
4.1. 304 Not Modified	18
4.2. 412 Precondition Failed	19
5. IANA Considerations	19
5.1. Status Code Registration	19
5.2. Header Field Registration	20
6. Security Considerations	20
7. Acknowledgments	20
8. References	20
8.1. Normative References	20
8.2. Informative References	21
Appendix A. Changes from RFC 2616	21
Appendix B. Collected ABNF	22
Appendix C. Change Log (to be removed by RFC Editor before publication)	22
C.1. Since RFC 2616	22
C.2. Since draft-ietf-httpbis-p4-conditional-00	22
C.3. Since draft-ietf-httpbis-p4-conditional-01	23
C.4. Since draft-ietf-httpbis-p4-conditional-02	23
C.5. Since draft-ietf-httpbis-p4-conditional-03	23
C.6. Since draft-ietf-httpbis-p4-conditional-04	23
C.7. Since draft-ietf-httpbis-p4-conditional-05	24
C.8. Since draft-ietf-httpbis-p4-conditional-06	24
C.9. Since draft-ietf-httpbis-p4-conditional-07	24
C.10. Since draft-ietf-httpbis-p4-conditional-08	24
C.11. Since draft-ietf-httpbis-p4-conditional-09	24
C.12. Since draft-ietf-httpbis-p4-conditional-10	24
C.13. Since draft-ietf-httpbis-p4-conditional-11	25
C.14. Since draft-ietf-httpbis-p4-conditional-12	25

C.15. Since draft-ietf-httpbis-p4-conditional-13	25
C.16. Since draft-ietf-httpbis-p4-conditional-14	25
Index	25

1. Introduction

This document defines the HTTP/1.1 conditional request mechanisms, including both response metadata that can be used to indicate or observe changes to resource state and request header fields that specify preconditions to be checked before performing the action given by the request method. 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. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

An implementation is not compliant if it fails to satisfy one or more of the "MUST" or "REQUIRED" level requirements for the protocols it implements. An implementation that satisfies all the "MUST" or "REQUIRED" level and all the "SHOULD" level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the "MUST" level requirements but not all the "SHOULD" level requirements for its protocols is said to be "conditionally compliant".

1.2. Syntax Notation

This specification uses the ABNF syntax defined in Section 1.2 of [Part1] (which extends the syntax defined in [RFC5234] with a list rule). Appendix B shows the collected ABNF, with the list rule expanded.

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

The ABNF rules below are defined in other parts:

```
quoted-string = <quoted-string, defined in [Part1], Section 1.2.2>
OWS           = <OWS, defined in [Part1], Section 1.2.2>
HTTP-date     = <HTTP-date, defined in [Part1], Section 6.1>
```

2. Resource State Metadata (Validators)

This specification defines two forms of metadata that are commonly used to observe resource state and test for preconditions: modification dates and opaque entity tags. 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. 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

2.1.1. 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.

2.1.2. 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 or If-Unmodified-Since header field, because the client has a cache entry 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.

2.2. 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 = quoted-string

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: ""
```

2.2.1. 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 stored 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.

2.2.2. Weak versus Strong

Since both origin servers and caches will compare two validators to decide if they indicate the same or different representations, one normally would expect that if the representation (including both representation header fields and representation body) changes in any way, then the associated validator would change as well. If this is true, then we call that validator a "strong validator". One example of a strong validator is an integer that is incremented in stable storage every time a representation is changed.

However, there might be cases when a server prefers to change the validator only when it desires cached representations to be invalidated. 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) 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). A validator that does not always change when the representation changes is a "weak validator".

One can think of a strong validator as part of an identifier for a specific representation, whereas a weak validator is part of an identifier for a set of equivalent representations (where this notion of equivalence is entirely governed by the origin server and beyond the scope of this specification).

An entity-tag is normally a strong validator, but the protocol provides a mechanism to tag an entity-tag as "weak".

A representation's modification time, if defined with only one-second resolution, could be a weak validator, since it is possible that the representation might be modified twice during a single second.

Support for weak validators is optional. However, weak validators allow for more efficient caching of equivalent objects; for example, a hit counter on a site is probably good enough if it is updated every few days or weeks, and any value during that period is likely "good enough" to be equivalent.

A strong entity-tag **MUST** change whenever the associated representation changes in any way. A weak entity-tag **SHOULD** change whenever the origin server considers prior representations to be unacceptable as a substitute for the current representation. In other words, a weak entity tag **SHOULD** change whenever the origin server wants caches to invalidate old responses.

A "strong entity-tag" **MAY** be shared by two representations of a resource only if they are equivalent by octet equality.

A "weak entity-tag", indicated by the "W/" prefix, **MAY** be shared by two representations of a resource. A weak entity-tag can only be used for weak comparison.

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 entity-tag **MUST** be unique across all versions of all representations associated with a particular resource over time. However, there is no implication of uniqueness across entity-tags of different resources (i.e., the same entity-tag value might be in use for representations of multiple resources at the same time and does not imply that those representations are equivalent).

2.2.3. 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.

A "use" of a validator is either when a client generates a request and includes the validator in a precondition, or when a server compares two validators.

Strong validators are usable in any context. Weak validators are only usable in contexts that do not depend on exact equality of a representation. For example, either kind is usable for a normal conditional GET.

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

An entity-tag is strong unless it is explicitly tagged as weak.

2.2.4. 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 should provide Last-Modified values. In those rare cases where the use of a Last-Modified value as a validator by an HTTP/1.0 system could result in a serious problem, then HTTP/1.1 origin servers should not provide one.

2.2.5. Example: Entity-tags varying on Content-Negotiated Resources

Consider a resource that is subject to content negotiation (Section 5 of [Part3]), and where the representations returned upon a GET request vary based on the Accept-Encoding request header field (Section 6.3 of [Part3]):

>> 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 must be distinct from an unencoded representation to prevent conflicts during cache updates and range requests. In contrast, transfer codings (Section 6.2 of [Part1]) apply only during message transfer and do not require distinct entity-tags.

3. Precondition Header Fields

This section defines the syntax and semantics of HTTP/1.1 header fields for applying preconditions on requests.

3.1. If-Match

The "If-Match" header field MAY 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

If any of the entity-tags listed in the If-Match field value match (as per Section 2.2.3) the entity-tag of the selected representation for the target resource, or if "*" is given and any current representation exists for the target resource, then the server MAY perform the request method as if the If-Match header field was not present.

If none of the entity-tags match, or if "*" is given and no current representation exists, the server MUST NOT perform the requested method. Instead, the server MUST respond with the 412 (Precondition

Failed) status code.

If the request would, without the If-Match header field, result in anything other than a 2xx or 412 status code, then the If-Match header field MUST be ignored.

Examples:

```
If-Match: "xyzzzy"
If-Match: "xyzzzy", "r2d2xxxx", "c3piozzzz"
If-Match: *
```

The result of a request having both an If-Match header field and either an If-None-Match or an If-Modified-Since header fields is undefined by this specification.

3.2. If-None-Match

The "If-None-Match" header field MAY 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 response if at least one of those representations matches the selected representation.

If-None-Match MAY 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

If any of the entity-tags listed in the If-None-Match field-value match (as per Section 2.2.3) the entity-tag of the selected representation, or if "*" is given and any current representation exists for that resource, then 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 none of the entity-tags match, then 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 or 304 status code, then the If-None-Match header field MUST be ignored. (See Section 2.2.4 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: *
```

The result of a request having both an If-None-Match header field and either an If-Match or an If-Unmodified-Since header fields is undefined by this specification.

3.3. If-Modified-Since

The "If-Modified-Since" header field MAY be used to make a request 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.

The result of a request having both an If-Modified-Since header field and either an If-Match or an If-Unmodified-Since header fields is undefined by this specification.

3.4. If-Unmodified-Since

The "If-Unmodified-Since" header field MAY 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 the request normally (i.e., without the If-Unmodified-Since header field) would result in anything other than a 2xx or 412 status code, the If-Unmodified-Since header field SHOULD be ignored.

If the specified date is invalid, the header field MUST be ignored.

The result of a request having both an If-Unmodified-Since header field and either an If-None-Match or an If-Modified-Since header fields is undefined by this specification.

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 9.3 of [Part1]) unless its omission is required by Section 9.3.1 of [Part1]. If a 200 response to the same request would have included any of the header fields Cache-Control, Content-Location, ETag, Expires, Last-Modified, 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 the outbound 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. IANA Considerations

5.1. 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

5.2. 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 2.2
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 2.1

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

6. Security Considerations

No additional security considerations have been identified beyond those applicable to HTTP in general [Part1].

7. Acknowledgments

8. References

8.1. Normative References

- [Part1] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing", draft-ietf-httpbis-p1-messaging-15 (work in progress), July 2011.
- [Part3] 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 3: Message Payload and Content Negotiation", draft-ietf-httpbis-p3-payload-15

(work in progress), July 2011.

- [Part5] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 5: Range Requests and Partial Responses", draft-ietf-httpbis-p5-range-15 (work in progress), July 2011.
- [Part6] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1, part 6: Caching", draft-ietf-httpbis-p6-cache-15 (work in progress), July 2011.
- [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.
- [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 2.2.2 and 3.2).

Change ABNF productions for header fields to only define the field value. (Section 3)

Appendix B. Collected ABNF

```
ETag = entity-tag

HTTP-date = <HTTP-date, defined in [Part1], Section 6.1>

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 1.2.2>

entity-tag = [ weak ] opaque-tag

opaque-tag = quoted-string

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

weak = %x57.2F ; W/

ABNF diagnostics:

; ETag defined but not used
; If-Match defined but not used
; If-Modified-Since defined but not used
; If-None-Match defined but not used
; If-Unmodified-Since defined but not used
; Last-Modified defined but not used
```

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

C.1. Since RFC 2616

Extracted relevant partitions from [RFC2616].

C.2. Since draft-ietf-httpbis-p4-conditional-00

Closed issues:

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

Other changes:

- o Move definitions of 304 and 412 condition codes from Part2.

C.3. Since draft-ietf-httpbis-p4-conditional-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.

C.4. Since draft-ietf-httpbis-p4-conditional-02

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/116>>: "Weak ETags on non-GET requests"

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.

C.5. Since draft-ietf-httpbis-p4-conditional-03

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/71>>: "Examples for ETag matching"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/124>>: "'entity value' undefined"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/126>>: "bogus 2068 Date header reference"

C.6. Since draft-ietf-httpbis-p4-conditional-04

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.

C.7. Since draft-ietf-httpbis-p4-conditional-05

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.

C.8. Since draft-ietf-httpbis-p4-conditional-06

Closed issues:

- o <http://tools.ietf.org/wg/httpbis/trac/ticket/153>: "case-sensitivity of etag weakness indicator"

C.9. Since draft-ietf-httpbis-p4-conditional-07

Closed issues:

- o <http://tools.ietf.org/wg/httpbis/trac/ticket/116>: "Weak ETags on non-GET requests" (If-Match still was defined to require strong matching)
- o <http://tools.ietf.org/wg/httpbis/trac/ticket/198>: "move IANA registrations for optional status codes"

C.10. Since draft-ietf-httpbis-p4-conditional-08

No significant changes.

C.11. Since draft-ietf-httpbis-p4-conditional-09

No significant changes.

C.12. Since draft-ietf-httpbis-p4-conditional-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/220>: "consider removing the 'changes from 2068' sections"

C.13. Since draft-ietf-httpbis-p4-conditional-11

None.

C.14. Since draft-ietf-httpbis-p4-conditional-12

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/224>>: "Header Classification"

C.15. Since draft-ietf-httpbis-p4-conditional-13

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/89>>: "If-* and entities"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/101>>: "Definition of validator weakness"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/276>>: "untangle ABNFs for header fields"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/269>>: "ETags and Quotes"

C.16. Since draft-ietf-httpbis-p4-conditional-14

None.

Index

3	304 Not Modified (status code)	18
4	412 Precondition Failed (status code)	19
E	ETag header field	8
G	Grammar	
	entity-tag	8
	ETag	8
	If-Match	14
	If-Modified-Since	16

- If-None-Match 15
- If-Unmodified-Since 18
- Last-Modified 6
- opaque-tag 8
- weak 8

H

Header Fields

- ETag 8
- If-Match 14
- If-Modified-Since 16
- If-None-Match 15
- If-Unmodified-Since 18
- Last-Modified 6

I

- If-Match header field 14
- If-Modified-Since header field 16
- If-None-Match header field 15
- If-Unmodified-Since header field 18

L

- Last-Modified header field 6

M

- metadata 6

S

- selected representation 5
- Status Codes
 - 304 Not Modified 18
 - 412 Precondition Failed 19

V

- validator 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/>

Jim Gettys
Alcatel-Lucent Bell Labs
21 Oak Knoll Road
Carlisle, MA 01741
USA

EMail: jg@freedesktop.org
URI: <http://gettys.wordpress.com/>

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

EMail: JeffMogul@acm.org

Henrik Frystyk Nielsen
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052
USA

EMail: henrikn@microsoft.com

Larry Masinter
Adobe Systems Incorporated
345 Park Ave
San Jose, CA 95110
USA

EMail: LMM@acm.org
URI: <http://larry.masinter.net/>

Paul J. Leach
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052

EMail: paulle@microsoft.com

Tim Berners-Lee
World Wide Web Consortium
MIT Computer Science and Artificial Intelligence Laboratory
The Stata Center, Building 32
32 Vassar Street
Cambridge, MA 02139
USA

Email: timbl@w3.org
URI: <http://www.w3.org/People/Berners-Lee/>

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

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

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

Phone: +49 251 2807760
Fax: +49 251 2807761
Email: julian.reschke@greenbytes.de
URI: <http://greenbytes.de/tech/webdav/>

HTTPbis Working Group
Internet-Draft
Obsoletes: 2616 (if approved)
Intended status: Standards Track
Expires: January 12, 2012

R. Fielding, Ed.
Adobe
J. Gettys
Alcatel-Lucent
J. Mogul
HP
H. Frystyk
Microsoft
L. Masinter
Adobe
P. Leach
Microsoft
T. Berners-Lee
W3C/MIT
Y. Lafon, Ed.
W3C
J. Reschke, Ed.
greenbytes
July 11, 2011

HTTP/1.1, part 5: Range Requests and Partial Responses
draft-ietf-httpbis-p5-range-15

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World Wide Web global information initiative since 1990. This document is Part 5 of the seven-part specification that defines the protocol referred to as "HTTP/1.1" and, taken together, obsoletes RFC 2616. Part 5 defines range-specific requests and the rules for constructing and combining responses to those requests.

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://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.16.

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 12, 2012.

Copyright Notice

Copyright (c) 2011 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	5
1.1. Requirements	5

1.2. Syntax Notation	5
1.2.1. Core Rules	6
1.2.2. ABNF Rules defined in other Parts of the Specification	6
2. Range Units	6
2.1. Range Specifier Registry	6
3. Status Code Definitions	7
3.1. 206 Partial Content	7
3.2. 416 Requested Range Not Satisfiable	8
4. Combining Ranges	8
5. Header Field Definitions	8
5.1. Accept-Ranges	9
5.2. Content-Range	9
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
8. Acknowledgments	16
9. References	17
9.1. Normative References	17
9.2. Informative References	17
Appendix A. Internet Media Type multipart/byteranges	17
Appendix B. Compatibility with Previous Versions	20
B.1. Changes from RFC 2616	20
Appendix C. Collected ABNF	20
Appendix D. Change Log (to be removed by RFC Editor before publication)	22
D.1. Since RFC 2616	22
D.2. Since draft-ietf-httpbis-p5-range-00	22
D.3. Since draft-ietf-httpbis-p5-range-01	22
D.4. Since draft-ietf-httpbis-p5-range-02	22
D.5. Since draft-ietf-httpbis-p5-range-03	23
D.6. Since draft-ietf-httpbis-p5-range-04	23
D.7. Since draft-ietf-httpbis-p5-range-05	23
D.8. Since draft-ietf-httpbis-p5-range-06	23
D.9. Since draft-ietf-httpbis-p5-range-07	24
D.10. Since draft-ietf-httpbis-p5-range-08	24
D.11. Since draft-ietf-httpbis-p5-range-09	24
D.12. Since draft-ietf-httpbis-p5-range-10	24
D.13. Since draft-ietf-httpbis-p5-range-11	24
D.14. Since draft-ietf-httpbis-p5-range-12	25
D.15. Since draft-ietf-httpbis-p5-range-13	25
D.16. Since draft-ietf-httpbis-p5-range-14	25

Index	25
-----------------	----

1. Introduction

HTTP clients often encounter interrupted data transfers as a result of cancelled requests or dropped connections. When a cache 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. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

An implementation is not compliant if it fails to satisfy one or more of the "MUST" or "REQUIRED" level requirements for the protocols it implements. An implementation that satisfies all the "MUST" or "REQUIRED" level and all the "SHOULD" level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the "MUST" level requirements but not all the "SHOULD" level requirements for its protocols is said to be "conditionally compliant".

1.2. Syntax Notation

This specification uses the ABNF syntax defined in Section 1.2 of [Part1] (which extends the syntax defined in [RFC5234] with a list rule). 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), LF (line feed), OCTET (any 8-bit sequence of data), SP (space), VCHAR (any visible USASCII character), and WSP (whitespace).

1.2.1. Core Rules

The core rules below are defined in Section 1.2.2 of [Part1]:

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

1.2.2. ABNF Rules defined in other Parts of the Specification

The ABNF rules below are defined in other parts:

HTTP-date = <HTTP-date, defined in [Part1], Section 6.1>

entity-tag = <entity-tag, defined in [Part4], Section 2.2>

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 are subject to IETF review ([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, Last-Modified, and/or Vary, if the header field would have been sent in a 200 response to the same request

If the 206 response is the result of an If-Range request, the response 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.

A cache MUST NOT combine a 206 response with other previously cached content if the ETag or Last-Modified header fields do not match exactly, see Section 4.

A cache that does not support the Range and Content-Range header fields MUST NOT cache 206 (Partial Content) responses. Furthermore, if a response uses a range unit that is not understood by the cache, then it MUST NOT be cached either.

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.

4. Combining Ranges

A response might transfer only a subrange of a representation, either because the request included one or more Range specifications, or because a connection closed prematurely. After several such transfers, a cache might have received several ranges of the same representation.

If a cache has a stored non-empty set of subranges for a representation, and an incoming response transfers another subrange, the cache MAY combine the new subrange with the existing set if both the following conditions are met:

- o Both the incoming response and the cache entry have a cache validator.
- o The two cache validators match using the strong comparison function (see Section 2.2.2 of [Part4]).

If either requirement is not met, the cache MUST use only the most recent partial response (based on the Date values transmitted with every response, and using the incoming response if these values are equal or missing), and MUST discard the other partial information.

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 = content-range-spec
```

```
content-range-spec      = 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 "*".

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

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 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 requests multiple ranges in one request, the server SHOULD return them in the order that they appeared in the request.

If the server ignores a byte-range-spec because it is syntactically invalid, the server SHOULD treat the request as if the invalid Range header field did not exist. (Normally, this means return a 200 response containing the full representation).

If the server receives a request (other than one including an If-Range header field) with an unsatisfiable Range header field (that is, all of whose byte-range-spec values have a first-byte-pos value greater than the current length of the selected resource), it SHOULD return a response code of 416 (Requested range not satisfiable) (Section 3.2).

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.

5.3. If-Range

If a client has a partial copy of a representation in its cache, and wishes to have an up-to-date copy of the entire representation in its cache, 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

Only a strong validator (Section 2.2.2 of [Part4]) is usable for range retrieval, since otherwise the client might end up with an internally inconsistent representation. Clients MUST NOT use weak validators in range requests. A cache or origin server receiving a conditional range request MUST use the strong comparison function to evaluate the condition.

If the client has no entity-tag for a representation, but does have a Last-Modified date, it MAY use that date in an If-Range header field. (The server can distinguish between a valid HTTP-date and any form of entity-tag by examining no more than two characters.) The If-Range header field SHOULD only be used together with a Range header field, and MUST be ignored if the request does not include a Range header field, or if the server does not support the sub-range operation.

If a client wishes to perform a sub-range retrieval on a value for which it has only a Last-Modified time and no opaque validator, it MAY do this only if the Last-Modified time is strong in the sense described in Section 2.1.2 of [Part4].

If the entity-tag given in the If-Range header field matches the current cache validator for the representation, then the server SHOULD provide the specified sub-range of the representation using a 206 (Partial Content) response. If the cache validator does not match, then the server SHOULD return 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.

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, HTTP/1.1 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	(this specification)

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].

8. Acknowledgments

Most of the specification of ranges is based on work originally done by Ari Luotonen and John Franks, with additional input from Steve Zilles, Daniel W. Connolly, Roy T. Fielding, Jim Gettys, Martin Hamilton, Koen Holtman, Shel Kaplan, Paul Leach, Alex Lopez-Ortiz, Larry Masinter, Jeff Mogul, Lou Montulli, David W. Morris, Luigi Rizzo, and Bill Weihl.

9. References

9.1. Normative References

- [Part1] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing", draft-ietf-httpbis-p1-messaging-15 (work in progress), July 2011.
- [Part4] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 4: Conditional Requests", draft-ietf-httpbis-p4-conditional-15 (work in progress), July 2011.
- [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].

Note: Despite the name "multipart/byteranges" is not limited to the byte ranges only.

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:

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

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--
```

Other example:

```
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 browsers 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. Compatibility with Previous Versions

B.1. Changes from RFC 2616

Clarify that it is not ok to use a weak cache 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. Collected ABNF

Accept-Ranges = acceptable-ranges

Content-Range = content-range-spec

HTTP-date = <HTTP-date, defined in [Part1], Section 6.1>

If-Range = entity-tag / HTTP-date

OWS = <OWS, defined in [Part1], Section 1.2.2>

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"

content-range-spec = byte-content-range-spec /
other-content-range-spec

entity-tag = <entity-tag, defined in [Part4], Section 2.2>

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 1.2.2>

ABNF diagnostics:

```
; Accept-Ranges defined but not used
; Content-Range defined but not used
; If-Range defined but not used
; Range defined but not used
```

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-p5-range-00

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/18>>: "Cache validators in 206 responses" (<http://purl.org/NET/http-errata#ifrange206>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/35>>: "Normative and Informative references"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/86>>: "Normative up-to-date references"

D.3. Since draft-ietf-httpbis-p5-range-01

Closed issues:

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

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.

D.4. Since draft-ietf-httpbis-p5-range-02

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 headers defined in this document.

D.5. Since draft-ietf-httpbis-p5-range-03

None.

D.6. Since draft-ietf-httpbis-p5-range-04

Closed issues:

- o <http://tools.ietf.org/wg/httpbis/trac/ticket/133>: "multipart/byteranges minimum number of parts"

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.

D.7. Since draft-ietf-httpbis-p5-range-05

Closed issues:

- o <http://tools.ietf.org/wg/httpbis/trac/ticket/142>: "State base for *-byte-pos and suffix-length"

Ongoing work on Custom Ranges

(<http://tools.ietf.org/wg/httpbis/trac/ticket/85>):

- o Remove bias in favor of byte ranges; allow custom ranges in ABNF.

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.

D.8. Since draft-ietf-httpbis-p5-range-06

Closed issues:

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

D.9. Since draft-ietf-httpbis-p5-range-07

Closed issues:

- o Fixed discrepancy in the If-Range definition about allowed validators.
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/150>>: "multipart/byteranges for custom range units"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/151>>: "range unit missing from other-ranges-specifier in Range header"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/198>>: "move IANA registrations for optional status codes"

D.10. Since draft-ietf-httpbis-p5-range-08

No significant changes.

D.11. Since draft-ietf-httpbis-p5-range-09

No significant changes.

D.12. Since draft-ietf-httpbis-p5-range-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/220>>: "consider removing the 'changes from 2068' sections"

Ongoing work on Custom Ranges

(<<http://tools.ietf.org/wg/httpbis/trac/ticket/85>>):

- o Add IANA registry.

D.13. Since draft-ietf-httpbis-p5-range-11

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/217>>: "Caches can't be required to serve ranges"

D.14. Since draft-ietf-httpbis-p5-range-12

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/224>>: "Header Classification"

D.15. Since draft-ietf-httpbis-p5-range-13

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/276>>: "untangle ABNFs for header fields"

D.16. Since draft-ietf-httpbis-p5-range-14

None.

Index

2	206 Partial Content (status code)	7
4	416 Requested Range Not Satisfiable (status code)	8
A	Accept-Ranges header field	9
C	Content-Range header field	9
G	Grammar	
	Accept-Ranges	9
	acceptable-ranges	9
	byte-content-range-spec	9
	byte-range-resp-spec	9
	byte-range-set	13
	byte-range-spec	13
	byte-ranges-specifier	13
	bytes-unit	6
	Content-Range	9
	content-range-spec	9
	first-byte-pos	13
	If-Range	12
	instance-length	9
	last-byte-pos	13

- other-range-unit 6
- Range 14
- range-unit 6
- ranges-specifier 13
- suffix-byte-range-spec 13
- suffix-length 13

H

Header Fields

- Accept-Ranges 9
- Content-Range 9
- If-Range 11
- Range 12

I

- If-Range header field 11

M

Media Type

- multipart/byteranges 17
- multipart/x-byteranges 20
- multipart/byteranges Media Type 17
- multipart/x-byteranges Media Type 20

R

- Range header field 12

S

Status Codes

- 206 Partial Content 7
- 416 Requested Range Not Satisfiable 8

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/>

Jim Gettys
Alcatel-Lucent Bell Labs
21 Oak Knoll Road
Carlisle, MA 01741
USA

EMail: jg@freedesktop.org
URI: <http://gettys.wordpress.com/>

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

EMail: JeffMogul@acm.org

Henrik Frystyk Nielsen
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052
USA

EMail: henrikn@microsoft.com

Larry Masinter
Adobe Systems Incorporated
345 Park Ave
San Jose, CA 95110
USA

EMail: LMM@acm.org
URI: <http://larry.masinter.net/>

Paul J. Leach
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052

EMail: paulle@microsoft.com

Tim Berners-Lee
World Wide Web Consortium
MIT Computer Science and Artificial Intelligence Laboratory
The Stata Center, Building 32
32 Vassar Street
Cambridge, MA 02139
USA

Email: timbl@w3.org
URI: <http://www.w3.org/People/Berners-Lee/>

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

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

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

Phone: +49 251 2807760
Fax: +49 251 2807761
Email: julian.reschke@greenbytes.de
URI: <http://greenbytes.de/tech/webdav/>

HTTPbis Working Group
Internet-Draft
Obsoletes: 2616 (if approved)
Intended status: Standards Track
Expires: January 12, 2012

R. Fielding, Ed.
Adobe
J. Gettys
Alcatel-Lucent
J. Mogul
HP
H. Frystyk
Microsoft
L. Masinter
Adobe
P. Leach
Microsoft
T. Berners-Lee
W3C/MIT
Y. Lafon, Ed.
W3C
M. Nottingham, Ed.

J. Reschke, Ed.
greenbytes
July 11, 2011

HTTP/1.1, part 6: Caching
draft-ietf-httpbis-p6-cache-15

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. This document is Part 6 of the seven-part specification that defines the protocol referred to as "HTTP/1.1" and, taken together, obsoletes RFC 2616. Part 6 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 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://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 C.16.

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 12, 2012.

Copyright Notice

Copyright (c) 2011 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	5
1.1. Purpose	5

1.2.	Terminology	5
1.3.	Requirements	7
1.4.	Syntax Notation	7
1.4.1.	Core Rules	7
1.4.2.	ABNF Rules defined in other Parts of the Specification	7
1.5.	Delta Seconds	8
2.	Cache Operation	8
2.1.	Response Cacheability	8
2.1.1.	Storing Partial and Incomplete Responses	9
2.2.	Constructing Responses from Caches	9
2.3.	Freshness Model	10
2.3.1.	Calculating Freshness Lifetime	11
2.3.2.	Calculating Age	12
2.3.3.	Serving Stale Responses	14
2.4.	Validation Model	14
2.5.	Request Methods that Invalidate	15
2.6.	Shared Caching of Authenticated Responses	16
2.7.	Caching Negotiated Responses	16
2.8.	Combining Responses	17
3.	Header Field Definitions	18
3.1.	Age	18
3.2.	Cache-Control	18
3.2.1.	Request Cache-Control Directives	19
3.2.2.	Response Cache-Control Directives	21
3.2.3.	Cache Control Extensions	23
3.3.	Expires	24
3.4.	Pragma	25
3.5.	Vary	26
3.6.	Warning	27
4.	History Lists	29
5.	IANA Considerations	29
5.1.	Cache Directive Registry	29
5.2.	Header Field Registration	30
6.	Security Considerations	30
7.	Acknowledgments	31
8.	References	31
8.1.	Normative References	31
8.2.	Informative References	32
Appendix A.	Changes from RFC 2616	32
Appendix B.	Collected ABNF	32
Appendix C.	Change Log (to be removed by RFC Editor before publication)	34
C.1.	Since RFC 2616	34
C.2.	Since draft-ietf-httpbis-p6-cache-00	34
C.3.	Since draft-ietf-httpbis-p6-cache-01	35
C.4.	Since draft-ietf-httpbis-p6-cache-02	35
C.5.	Since draft-ietf-httpbis-p6-cache-03	35

C.6.	Since draft-ietf-httpbis-p6-cache-04	35
C.7.	Since draft-ietf-httpbis-p6-cache-05	36
C.8.	Since draft-ietf-httpbis-p6-cache-06	36
C.9.	Since draft-ietf-httpbis-p6-cache-07	36
C.10.	Since draft-ietf-httpbis-p6-cache-08	37
C.11.	Since draft-ietf-httpbis-p6-cache-09	37
C.12.	Since draft-ietf-httpbis-p6-cache-10	38
C.13.	Since draft-ietf-httpbis-p6-cache-11	38
C.14.	Since draft-ietf-httpbis-p6-cache-12	38
C.15.	Since draft-ietf-httpbis-p6-cache-13	38
C.16.	Since draft-ietf-httpbis-p6-cache-14	38
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.

Caching would be useless if it did not significantly improve performance. The goal of caching in HTTP/1.1 is to reuse a prior response message to satisfy a current request. In some cases, a stored response can be reused without the need for a network request, reducing latency and network round-trips; a "freshness" mechanism is used for this purpose (see Section 2.3). Even when a new request is required, it is often possible to reuse all or parts of the payload of a prior response to satisfy the request, thereby reducing network bandwidth usage; a "validation" mechanism is used for this purpose (see Section 2.4).

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 is accessible to 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.

1.3. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

An implementation is not compliant if it fails to satisfy one or more of the "MUST" or "REQUIRED" level requirements for the protocols it implements. An implementation that satisfies all the "MUST" or "REQUIRED" level and all the "SHOULD" level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the "MUST" level requirements but not all the "SHOULD" level requirements for its protocols is said to be "conditionally compliant".

1.4. Syntax Notation

This specification uses the ABNF syntax defined in Section 1.2 of [Part1] (which extends the syntax defined in [RFC5234] with a list rule). Appendix B shows the collected ABNF, with the list rule expanded.

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

1.4.1. Core Rules

The core rules below are defined in Section 1.2.2 of [Part1]:

```
quoted-string = <quoted-string, defined in [Part1], Section 1.2.2>
token         = <token, defined in [Part1], Section 1.2.2>
OWS          = <OWS, defined in [Part1], Section 1.2.2>
```

1.4.2. ABNF Rules defined in other Parts of the Specification

The ABNF rules below are defined in other parts:

```
field-name    = <field-name, defined in [Part1], Section 3.2>
HTTP-date     = <HTTP-date, defined in [Part1], Section 6.1>
port          = <port, defined in [Part1], Section 2.7>
pseudonym     = <pseudonym, defined in [Part1], Section 9.9>
uri-host      = <uri-host, defined in [Part1], Section 2.7>
```

1.5. 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 **SHOULD** 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. Cache Operation

2.1. Response Cacheability

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 3.2) does not appear in request or response header fields, and
- o the "private" cache response directive (see Section 3.2.2) 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 2.6), and
- o the response either:
 - * contains an Expires header field (see Section 3.3), or
 - * contains a max-age response cache directive (see Section 3.2.2), or
 - * contains a s-maxage response cache directive and the cache is shared, or
 - * contains a Cache Control Extension (see Section 3.2.3) that allows it to be cached, or

- * has a status code that can be served with heuristic freshness (see Section 2.3.1.1).

Note that any of the requirements listed above can be overridden by a cache-control extension; see Section 3.2.3.

In this context, a cache has "understood" a request method or a response status code if it recognises it and implements any cache-specific behavior. In particular, 206 Partial Content responses cannot be cached by an implementation that does not handle partial content (see Section 2.1.1).

Note that in normal operation, most 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.

2.1.1. Storing Partial and Incomplete Responses

A cache that receives an incomplete response (for example, with fewer bytes of data than specified in a Content-Length header field) can store the response, but MUST treat it as a partial response [Part5]. Partial responses can be combined as described in Section 4 of [Part5]; the result might be a full response or might still be partial. A cache MUST NOT return a partial response to a client without explicitly marking it as such using the 206 (Partial Content) status code.

A cache that does not support the Range and Content-Range header fields MUST NOT store incomplete or partial responses.

2.2. Constructing Responses from Caches

For a presented request, a cache MUST NOT return a stored response, unless:

- o The presented effective request URI (Section 4.3 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 2.7), and
- o the presented request and stored response are free from directives that would prevent its use (see Section 3.2 and Section 3.4), and

- o the stored response is either:
 - * fresh (see Section 2.3), or
 - * allowed to be served stale (see Section 2.3.3), or
 - * successfully validated (see Section 2.4).

Note that any of the requirements listed above can be overridden by a cache-control extension; see Section 3.2.3.

When a stored response is used to satisfy a request without validation, a cache MUST include a single Age header field (Section 3.1) in the response with a value equal to the stored response's `current_age`; see Section 2.3.2.

A cache MUST write through requests with methods that are unsafe (Section 7.1.1 of [Part2]) to the origin server; i.e., a cache must not 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 2.5.

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.

2.3. 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 3.3) or the max-age response cache directive (Section 3.2.2). 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 2.3.3).

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 2.3.1; the `current_age` is defined in Section 2.3.2.

Additionally, clients might need to influence freshness calculation. They can do this using several request cache directives, with the effect of either increasing or loosening constraints on freshness. See Section 3.2.1.

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 4 for an explanation of the difference between caches and history mechanisms.

2.3.1. 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 3.2.2) is present, use its value, or
- o If the `max-age` response cache directive (Section 3.2.2) is present, use its value, or
- o If the `Expires` response header field (Section 3.3) 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 2.3.1.1.

Note that this calculation is not vulnerable to clock skew, since all of the information comes from the origin server.

2.3.1.1. 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 8 of [Part2]: 200, 203, 206, 300, 301 and 410), 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 2.1 of [Part4]), a cache SHOULD NOT use a heuristic expiration value that is more than some fraction of the interval since that time. A typical setting of this fraction might be 10%.

Note: RFC 2616 ([RFC2616], Section 13.9) required that caches do not calculate 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.

2.3.2. 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 3.1), 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 9.3 of [Part1] 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 are combined as

```
corrected_initial_age = max(apparent_age, corrected_age_value);
```

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;
```

2.3.3. 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 2.3.

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 3.2.2).

A cache **SHOULD 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 3.2.1).

A cache **SHOULD** append a Warning header field with the 110 warn-code (see Section 3.6) 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 **SHOULD** forward it to the requesting client without adding a new Warning (but without removing any existing Warning header fields). A cache **SHOULD NOT** attempt to validate a response simply because that response became stale in transit.

2.4. 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 2.7), 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 SHOULD add an If-Modified-Since header field whose value is that of the Last-Modified header field from the selected (see Section 2.7) stored response, if available.

Additionally, a cache SHOULD 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 SHOULD NOT 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.

A 304 (Not Modified) response status code indicates that the stored response can be updated and reused; see Section 2.8.

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, a cache SHOULD use the full response to satisfy the request and MAY replace the stored response.

If a cache receives a 5xx response while attempting to validate a response, it MAY either forward this response to the requesting client, or act as if the server failed to respond. In the latter case, it MAY return a previously stored response (see Section 2.3.3).

2.5. Request Methods that Invalidate

Because unsafe request methods (Section 7.1.1 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 4.3 of [Part1]) as well as the URI(s) in the Location and Content-Location 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 header field if the host part of that URI differs from the host part in the effective request URI (Section 4.3 of [Part1]). This helps prevent denial of service attacks.

A cache SHOULD invalidate the effective request URI (Section 4.3 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 or 3xx 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.

2.6. Shared Caching of Authenticated Responses

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 3.2.2) 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 2.3.3) 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.

2.7. Caching Negotiated Responses

When a cache receives a request that can be satisfied by a stored response that has a Vary header field (Section 3.5), 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 2.2.

If no selected response is available, the cache MAY forward the presented request to the origin server in a conditional request; see Section 2.4.

2.8. Combining Responses

When a cache receives a 304 (Not Modified) response or a 206 (Partial Content) response (in this section, the "new" response), it needs to create an updated response by combining the stored response with the new one, so that the updated response can be used to satisfy the request, and potentially update the cached response.

If the new response contains an ETag, it identifies the stored response to use. [[TODO-mention-CL: might need language about Content-Location here]][TODO-select-for-combine: Shouldn't this be the selected response?]]

When the new response's status code is 206 (partial content), a cache MUST NOT combine it with the old response if either response does not have a validator, and MUST NOT combine it with the old response when those validators do not match with the strong comparison function (see Section 2.2.2 of [Part4]).

The stored response header fields are used as those of the updated response, except that

- o a cache MUST delete any stored Warning header fields with warn-code 1xx (see Section 3.6).
- o a cache MUST retain any stored Warning header fields with warn-code 2xx.
- o a cache MUST use other header fields provided in the new response to replace all instances of the corresponding header fields from the stored response.

A cache MUST use the updated response header fields to replace those of the stored response (unless the stored response is removed). In the case of a 206 response, a cache MAY store the combined representation.

3. Header Field Definitions

This section defines the syntax and semantics of HTTP/1.1 header fields related to caching.

3.1. 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 2.3.2.

Age = delta-seconds

Age field-values are non-negative integers, representing time in seconds (see Section 1.5).

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.

3.2. 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 3.2.3 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 3.4).

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-Control    = 1#cache-directive
```

```
cache-directive = cache-request-directive  
                  / cache-response-directive
```

```
cache-extension = token [ "=" ( token / quoted-string ) ]
```

3.2.1. Request Cache-Control Directives

```
cache-request-directive =  
    "no-cache"  
    / "no-store"  
    / "max-age" "=" delta-seconds  
    / "max-stale" [ "=" delta-seconds ]  
    / "min-fresh" "=" delta-seconds  
    / "no-transform"  
    / "only-if-cached"  
    / cache-extension
```

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.

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.

max-age

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.

max-stale

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.

min-fresh

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.

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.

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.

3.2.2. Response Cache-Control Directives

```
cache-response-directive =  
    "public"  
    / "private" [ "=" DQUOTE 1#field-name DQUOTE ]  
    / "no-cache" [ "=" DQUOTE 1#field-name DQUOTE ]  
    / "no-store"  
    / "no-transform"  
    / "must-revalidate"  
    / "proxy-revalidate"  
    / "max-age" "=" delta-seconds  
    / "s-maxage" "=" delta-seconds  
    / cache-extension
```

public

The public response directive indicates that a response whose associated request contains an 'Authentication' header MAY be stored (see Section 2.6).

private

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.

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.

no-cache

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, this requirement is limited to the field-values associated with the listed response header fields. That is, a cache MUST NOT send the specified field-name(s) in the response to a subsequent request without successful validation on 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.

Note: Most 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.

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.

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.

A server SHOULD send the must-revalidate directive if and only if failure to validate a request on the representation could result in incorrect operation, such as a silently unexecuted financial transaction.

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.

max-age

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.

s-maxage

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.

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.

3.2.3. 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 may 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).

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 are subject to IETF review ([RFC5226], Section 4.1).

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

3.3. Expires

The "Expires" header field gives the date/time after which the response is considered stale. See Section 2.3 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 6.1 of [Part1]; a sender MUST use the rfc1123-date format.

Expires = HTTP-date

For example

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

Note: If a response includes a Cache-Control field with the max-age directive (see Section 3.2.2), that directive overrides the Expires field. Likewise, the s-maxage directive overrides Expires in shared caches.

A server SHOULD NOT send Expires dates more than one year in the future.

A cache MUST treat other invalid date formats, especially including the value "0", as in the past (i.e., "already expired").

3.4. 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 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 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 3.2.1).

When sending a no-cache request, a client SHOULD include both 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.

3.5. 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 2.7. determines, while the response is fresh, whether a cache is permitted to use the response to reply to a subsequent request without validation; see Section 2.7.

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.

3.6. 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 SHOULD order them with this user agent behavior in mind. New Warning header fields SHOULD be added after any existing Warning headers 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.

110 Response is stale

A cache SHOULD include this whenever the returned response is stale.

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.

112 Disconnected operation

A cache SHOULD include this if it is intentionally disconnected from the rest of the network for a period of time.

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.

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.

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.

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.

4. 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 2.3) 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).

5. IANA Considerations

5.1. Cache Directive Registry

The registration procedure for HTTP Cache Directives is defined by Section 3.2.3 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 3.2.1, Section 3.2.2
max-stale	Section 3.2.1
min-fresh	Section 3.2.1
must-revalidate	Section 3.2.2
no-cache	Section 3.2.1, Section 3.2.2
no-store	Section 3.2.1, Section 3.2.2
no-transform	Section 3.2.1, Section 3.2.2
only-if-cached	Section 3.2.1
private	Section 3.2.2
proxy-revalidate	Section 3.2.2
public	Section 3.2.2
s-maxage	Section 3.2.2
stale-if-error	[RFC5861], Section 4
stale-while-revalidate	[RFC5861], Section 3

5.2. 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
Age	http	standard	Section 3.1
Cache-Control	http	standard	Section 3.2
Expires	http	standard	Section 3.3
Pragma	http	standard	Section 3.4
Vary	http	standard	Section 3.5
Warning	http	standard	Section 3.6

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

6. 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.

7. Acknowledgments

Much of the content and presentation of the caching design is due to suggestions and comments from individuals including: Shel Kaphan, Paul Leach, Koen Holtman, David Morris, and Larry Masinter.

8. References

8.1. Normative References

- [Part1] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing", draft-ietf-httpbis-p1-messaging-15 (work in progress), July 2011.
- [Part2] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 2: Message Semantics", draft-ietf-httpbis-p2-semantics-15 (work in progress), July 2011.
- [Part4] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 4: Conditional Requests", draft-ietf-httpbis-p4-conditional-15 (work in progress), July 2011.
- [Part5] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 5: Range Requests and Partial Responses", draft-ietf-httpbis-p5-range-15 (work in progress), July 2011.
- [Part7] 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-15 (work in progress), July 2011.
- [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

- [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 2.3.2)

Remove requirement to consider Content-Location in successful responses in order to determine the appropriate response to use.
(Section 2.4)

Clarify denial of service attack avoidance requirement.
(Section 2.5)

Change ABNF productions for header fields to only define the field value. (Section 3)

Do not mention RFC 2047 encoding and multiple languages in Warning header fields anymore, as these aspects never were implemented.
(Section 3.6)

Appendix B. Collected ABNF

Age = delta-seconds

Cache-Control = *("," OWS) cache-directive *(OWS "," [OWS
cache-directive])

Expires = HTTP-date

```

HTTP-date = <HTTP-date, defined in [Part1], Section 6.1>

OWS = <OWS, defined in [Part1], Section 1.2.2>

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 = cache-request-directive / cache-response-directive
cache-extension = token [ "=" ( token / quoted-string ) ]
cache-request-directive = "no-cache" / "no-store" / ( "max-age="
  delta-seconds ) / ( "max-stale" [ "=" delta-seconds ] ) / (
  "min-fresh=" delta-seconds ) / "no-transform" / "only-if-cached" /
  cache-extension
cache-response-directive = "public" / ( "private" [ "=" DQUOTE *( ","
  OWS ) field-name *( OWS "," [ OWS field-name ] ) DQUOTE ] ) / (
  "no-cache" [ "=" DQUOTE *( "," OWS ) field-name *( OWS "," [ OWS
  field-name ] ) DQUOTE ] ) / "no-store" / "no-transform" /
  "must-revalidate" / "proxy-revalidate" / ( "max-age=" delta-seconds
  ) / ( "s-maxage=" delta-seconds ) / cache-extension

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 2.7>
pragma-directive = "no-cache" / extension-pragma
pseudonym = <pseudonym, defined in [Part1], Section 9.9>

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

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

uri-host = <uri-host, defined in [Part1], Section 2.7>

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
  ]

```

ABNF diagnostics:

```
; Age defined but not used
; Cache-Control defined but not used
; Expires defined but not used
; Pragma defined but not used
; Vary defined but not used
; Warning defined but not used
```

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

C.1. Since RFC 2616

Extracted relevant partitions from [RFC2616].

C.2. Since draft-ietf-httpbis-p6-cache-00

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/9>>: "Trailer" (<http://purl.org/NET/http-errata#trailer-hop>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/12>>: "Invalidation after Update or Delete" (<http://purl.org/NET/http-errata#invalidupd>)
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/35>>: "Normative and Informative references"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/48>>: "Date reference typo"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/49>>: "Connection header text"
- 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"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/87>>: "typo in 13.2.2"

Other changes:

- o Use names of RFC4234 core rules DQUOTE and HTAB (work in progress on <http://tools.ietf.org/wg/httpbis/trac/ticket/36>)

C.3. Since draft-ietf-httpbis-p6-cache-01

Closed issues:

- o <http://tools.ietf.org/wg/httpbis/trac/ticket/82>: "rel_path not used"

Other changes:

- o Get rid of duplicate BNF rule names ("host" -> "uri-host") (work in progress on <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.

C.4. Since draft-ietf-httpbis-p6-cache-02

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.

C.5. Since draft-ietf-httpbis-p6-cache-03

Closed issues:

- o <http://tools.ietf.org/wg/httpbis/trac/ticket/106>: "Vary header classification"

C.6. Since draft-ietf-httpbis-p6-cache-04

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.

C.7. Since draft-ietf-httpbis-p6-cache-05

This is a total rewrite of this part of the specification.

Affected issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/54>>: "Definition of lxx Warn-Codes"
- 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/138>>: "The role of Warning and Semantic Transparency in Caching"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/139>>: "Methods and Caching"

In addition: 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.

C.8. Since draft-ietf-httpbis-p6-cache-06

Closed issues:

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

Affected issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/37>>: "Vary and non-existent headers"

C.9. Since draft-ietf-httpbis-p6-cache-07

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/54>>: "Definition of lxx Warn-Codes"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/167>>: "Content-Location on 304 responses"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/169>>: "private and no-cache CC directives with headers"

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/187>>: "RFC2047 and warn-text"

C.10. Since draft-ietf-httpbis-p6-cache-08

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/147>>: "serving negotiated responses from cache: header-specific canonicalization"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/197>>: "Effect of CC directives on history lists"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/291>>: "Cache Extensions can override no-store, etc."

Affected issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/199>>: Status codes and caching

Partly resolved issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/60>>: "Placement of 13.5.1 and 13.5.2"

C.11. Since draft-ietf-httpbis-p6-cache-09

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/29>>: "Age calculation"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/168>>: "Clarify differences between / requirements for request and response CC directives"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/174>>: "Caching authenticated responses"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/208>>: "IANA registry for cache-control directives"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/211>>: "Heuristic caching of URLs with query components"

Partly resolved issues:

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

C.12. Since draft-ietf-httpbis-p6-cache-10

Closed issues:

- 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"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/223>>: "Allowing heuristic caching for new status codes"
- o Clean up TODOs and prose in "Combining Responses."

C.13. Since draft-ietf-httpbis-p6-cache-11

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/204>>: "Text about clock requirement for caches belongs in p6"

C.14. Since draft-ietf-httpbis-p6-cache-12

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/224>>: "Header Classification"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/268>>: "Clarify 'public' "

C.15. Since draft-ietf-httpbis-p6-cache-13

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/276>>: "untangle ABNFs for header fields"

C.16. Since draft-ietf-httpbis-p6-cache-14

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/38>>: "Mismatch Vary"

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/235>>: "Cache Invalidation only happens upon successful responses"
- 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/289>>: "Proxies don't 'understand' methods"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/292>>: "Pragma"

Index

A

age 6
Age header field 18

C

cache 5
Cache Directives
 max-age 20, 23
 max-stale 20
 min-fresh 20
 must-revalidate 22
 no-cache 19, 21
 no-store 19, 22
 no-transform 20, 23
 only-if-cached 20
 private 21
 proxy-revalidate 23
 public 21
 s-maxage 23
Cache-Control header field 18
cacheable 5

E

Expires header field 24
explicit expiration time 6

F

first-hand 6
fresh 6
freshness lifetime 6

G

Grammar
 Age 18
 Cache-Control 19

- cache-extension 19
- cache-request-directive 19
- cache-response-directive 21
- delta-seconds 8
- Expires 25
- extension-pragma 25
- Pragma 25
- pragma-directive 25
- Vary 26
- warn-agent 27
- warn-code 27
- warn-date 27
- warn-text 27
- Warning 27
- warning-value 27

H

- Header Fields
 - Age 18
 - Cache-Control 18
 - Expires 24
 - Pragma 25
 - Vary 26
 - Warning 27
- heuristic expiration time 6

M

- max-age
 - Cache Directive 20, 23
- max-stale
 - Cache Directive 20
- min-fresh
 - Cache Directive 20
- must-revalidate
 - Cache Directive 22

N

- no-cache
 - Cache Directive 19, 21
- no-store
 - Cache Directive 19, 22
- no-transform
 - Cache Directive 20, 23

O

- only-if-cached
 - Cache Directive 20

P	
Pragma header field	25
private	
Cache Directive	21
private cache	5
proxy-revalidate	
Cache Directive	23
public	
Cache Directive	21
S	
s-maxage	
Cache Directive	23
shared cache	5
stale	6
V	
validator	6
Vary header field	26
W	
Warning header field	27

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/>

Jim Gettys
Alcatel-Lucent Bell Labs
21 Oak Knoll Road
Carlisle, MA 01741
USA

EMail: jg@freedesktop.org
URI: <http://gettys.wordpress.com/>

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

EMail: JeffMogul@acm.org

Henrik Frystyk Nielsen
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052
USA

EMail: henrikn@microsoft.com

Larry Masinter
Adobe Systems Incorporated
345 Park Ave
San Jose, CA 95110
USA

EMail: LMM@acm.org
URI: <http://larry.masinter.net/>

Paul J. Leach
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052

EMail: paulle@microsoft.com

Tim Berners-Lee
World Wide Web Consortium
MIT Computer Science and Artificial Intelligence Laboratory
The Stata Center, Building 32
32 Vassar Street
Cambridge, MA 02139
USA

EMail: timbl@w3.org
URI: <http://www.w3.org/People/Berners-Lee/>

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

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

Mark Nottingham (editor)

EMail: mnot@mnot.net
URI: <http://www.mnot.net/>

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

Phone: +49 251 2807760
Fax: +49 251 2807761
EMail: julian.reschke@greenbytes.de
URI: <http://greenbytes.de/tech/webdav/>

HTTPbis Working Group
Internet-Draft
Obsoletes: 2616 (if approved)
Updates: 2617 (if approved)
Intended status: Standards Track
Expires: January 12, 2012

R. Fielding, Ed.
Adobe
J. Gettys
Alcatel-Lucent
J. Mogul
HP
H. Frystyk
Microsoft
L. Masinter
Adobe
P. Leach
Microsoft
T. Berners-Lee
W3C/MIT
Y. Lafon, Ed.
W3C
J. Reschke, Ed.
greenbytes
July 11, 2011

HTTP/1.1, part 7: Authentication
draft-ietf-httpbis-p7-auth-15

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World Wide Web global information initiative since 1990. This document is Part 7 of the seven-part specification that defines the protocol referred to as "HTTP/1.1" and, taken together, obsoletes RFC 2616. Part 7 defines HTTP Authentication.

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://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 C.16.

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 12, 2012.

Copyright Notice

Copyright (c) 2011 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. Requirements	4
1.2. Syntax Notation	4
1.2.1. Core Rules	4
2. Access Authentication Framework	5
2.1. Authentication Scheme Registry	7
3. Status Code Definitions	7
3.1. 401 Unauthorized	7
3.2. 407 Proxy Authentication Required	7
4. Header Field Definitions	8
4.1. Authorization	8
4.2. Proxy-Authenticate	9
4.3. Proxy-Authorization	9
4.4. WWW-Authenticate	9
5. IANA Considerations	10
5.1. Authentication Scheme Registry	10
5.2. Status Code Registration	10
5.3. Header Field Registration	10
6. Security Considerations	10
6.1. Authentication Credentials and Idle Clients	11
7. Acknowledgments	11
8. References	11
8.1. Normative References	11
8.2. Informative References	12
Appendix A. Changes from RFC 2616	12
Appendix B. Collected ABNF	13
Appendix C. Change Log (to be removed by RFC Editor before publication)	13
C.1. Since RFC 2616	13
C.2. Since draft-ietf-httpbis-p7-auth-00	13
C.3. Since draft-ietf-httpbis-p7-auth-01	14
C.4. Since draft-ietf-httpbis-p7-auth-02	14
C.5. Since draft-ietf-httpbis-p7-auth-03	14
C.6. Since draft-ietf-httpbis-p7-auth-04	14
C.7. Since draft-ietf-httpbis-p7-auth-05	14
C.8. Since draft-ietf-httpbis-p7-auth-06	14
C.9. Since draft-ietf-httpbis-p7-auth-07	15
C.10. Since draft-ietf-httpbis-p7-auth-08	15
C.11. Since draft-ietf-httpbis-p7-auth-09	15
C.12. Since draft-ietf-httpbis-p7-auth-10	15
C.13. Since draft-ietf-httpbis-p7-auth-11	15
C.14. Since draft-ietf-httpbis-p7-auth-12	15
C.15. Since draft-ietf-httpbis-p7-auth-13	16
C.16. Since draft-ietf-httpbis-p7-auth-14	16
Index	16

1. Introduction

This document defines HTTP/1.1 access control and authentication. It includes the relevant parts of RFC 2616 with only minor changes, 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. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

An implementation is not compliant if it fails to satisfy one or more of the "MUST" or "REQUIRED" level requirements for the protocols it implements. An implementation that satisfies all the "MUST" or "REQUIRED" level and all the "SHOULD" level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the "MUST" level requirements but not all the "SHOULD" level requirements for its protocols is said to be "conditionally compliant".

1.2. Syntax Notation

This specification uses the ABNF syntax defined in Section 1.2 of [Part1] (which extends the syntax defined in [RFC5234] with a list rule). Appendix B shows the collected ABNF, with the list rule expanded.

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

1.2.1. Core Rules

The core rules below are defined in Section 1.2.2 of [Part1]:

quoted-string = <quoted-string, defined in [Part1], Section 1.2.2>
token = <token, defined in [Part1], Section 1.2.2>
OWS = <OWS, defined in [Part1], Section 1.2.2>

2. Access Authentication Framework

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 a comma-separated list of attribute-value pairs which carry the parameters necessary for achieving authentication via that scheme.

auth-scheme = token
auth-param = token "=" (token / quoted-string)

The 401 (Unauthorized) response message is used by an origin server to challenge the authorization of a user agent. This response MUST include a WWW-Authenticate header field containing 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 1#auth-param

Note: User agents will need to take special care in parsing the WWW-Authenticate or Proxy-Authenticate header field value if it contains more than one challenge, or if more than one WWW-Authenticate header field is provided, since the contents of a challenge can itself contain a comma-separated list of authentication parameters.

Note: Many browsers fail to parse challenges containing unknown schemes. A workaround for this problem is to list well-supported schemes (such as "basic") first.

The authentication parameter realm is defined for all authentication schemes:

realm = "realm" "=" realm-value
realm-value = quoted-string

The realm directive (case-insensitive) is required for all authentication schemes that issue a challenge. The realm value (case-sensitive), in combination with the canonical root URI (the

scheme and authority components of the effective request URI; see Section 4.3 of [Part1]) of the server being accessed, defines the protection space. These realms allow the protected resources on a server to be partitioned into a set of protection spaces, each with its own authentication scheme and/or authorization database. The realm value is a string, generally assigned by the origin server, which can have additional semantics specific to the authentication scheme. Note that there can be multiple challenges with the same auth-scheme but different realms.

A user agent that wishes to authenticate itself with an origin server -- usually, but not necessarily, after receiving a 401 (Unauthorized) -- MAY 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) -- MAY do so by including a Proxy-Authorization header field with the request. Both the Authorization field value and the Proxy-Authorization field value consist of credentials containing the authentication information of the client for the realm of the resource being requested. The user agent MUST choose to use one of the challenges with the strongest auth-scheme it understands and request credentials from the user based upon that challenge.

```
credentials = auth-scheme ( token
                             / quoted-string
                             / #auth-param )
```

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.

If the origin server does not wish to accept the credentials sent with a request, it SHOULD return a 401 (Unauthorized) response. The response MUST include a WWW-Authenticate header field containing at least one (possibly new) challenge applicable to the requested resource. If a proxy does not accept the credentials sent with a request, it SHOULD return a 407 (Proxy Authentication Required). The response MUST include a Proxy-Authenticate header field containing a (possibly new) challenge applicable to the proxy for the requested resource.

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 headers unmodified and follow the rules found in Section 4.1.

2.1. 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

Values to be added to this name space are subject to IETF review ([RFC5226], Section 4.1).

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

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 a challenge that indicates the authentication scheme and parameters applicable to the proxy for this effective request URI (Section 4.3 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 SHOULD NOT be passed on 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.

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 4.3 of [Part1]). It MUST be included in 401 (Unauthorized) response messages.

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.

5. IANA Considerations

5.1. Authentication Scheme Registry

The registration procedure for HTTP Authentication Schemes is defined by Section 2.1 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.

7. Acknowledgments

This specification takes over the definition of the HTTP Authentication Framework, previously defined in RFC 2617. We thank to 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.

[[acks: HTTPbis acknowledgements.]]

8. References

8.1. Normative References

[Part1] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing", draft-ietf-httpbis-pl-messaging-15

(work in progress), July 2011.

- [Part6] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1, part 6: Caching", draft-ietf-httpbis-p6-cache-15 (work in progress), July 2011.
- [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.
- [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 RFC 2616

Change ABNF productions for header fields to only define the field value. (Section 4)

Appendix B. Collected ABNF

```
Authorization = credentials

OWS = <OWS, defined in [Part1], Section 1.2.2>

Proxy-Authenticate = *( "," OWS ) challenge *( OWS "," [ OWS
  challenge ] )
Proxy-Authorization = credentials

WWW-Authenticate = *( "," OWS ) challenge *( OWS "," [ OWS challenge
  ] )

auth-param = token "=" ( token / quoted-string )
auth-scheme = token

challenge = auth-scheme 1*SP *( "," OWS ) auth-param *( OWS "," [ OWS
  auth-param ] )
credentials = auth-scheme ( token / quoted-string / [ ( "," /
  auth-param ) *( OWS "," [ OWS auth-param ] ) ] )

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

realm = "realm=" realm-value
realm-value = quoted-string

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

ABNF diagnostics:

; Authorization defined but not used
; Proxy-Authenticate defined but not used
; Proxy-Authorization defined but not used
; WWW-Authenticate defined but not used
; realm defined but not used
```

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

C.1. Since RFC 2616

Extracted relevant partitions from [RFC2616].

C.2. Since draft-ietf-httpbis-p7-auth-00

Closed issues:

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

C.3. Since draft-ietf-httpbis-p7-auth-01

Ongoing work on ABNF conversion

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

- o Explicitly import BNF rules for "challenge" and "credentials" from RFC2617.
- o Add explicit references to BNF syntax and rules imported from other parts of the specification.

C.4. Since draft-ietf-httpbis-p7-auth-02

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.

C.5. Since draft-ietf-httpbis-p7-auth-03

C.6. Since draft-ietf-httpbis-p7-auth-04

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.

C.7. Since draft-ietf-httpbis-p7-auth-05

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.

C.8. Since draft-ietf-httpbis-p7-auth-06

None.

C.9. Since draft-ietf-httpbis-p7-auth-07

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/198>>: "move IANA registrations for optional status codes"

C.10. Since draft-ietf-httpbis-p7-auth-08

No significant changes.

C.11. Since draft-ietf-httpbis-p7-auth-09

Partly resolved issues:

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

C.12. Since draft-ietf-httpbis-p7-auth-10

None.

C.13. Since draft-ietf-httpbis-p7-auth-11

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/130>>: "introduction to part 7 is work-in-progress"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/195>>: "auth-param syntax"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/224>>: "Header Classification"
- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/237>>: "absorbing the auth framework from 2617"

Partly resolved issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/141>>: "should we have an auth scheme registry"

C.14. Since draft-ietf-httpbis-p7-auth-12

None.

C.15. Since draft-ietf-httpbis-p7-auth-13

Closed issues:

- o <<http://tools.ietf.org/wg/httpbis/trac/ticket/276>>: "untangle ABNFs for header fields"

C.16. Since draft-ietf-httpbis-p7-auth-14

None.

Index

4	
401 Unauthorized (status code)	7
407 Proxy Authentication Required (status code)	7
A	
auth-param	5
auth-scheme	5
Authorization header field	8
C	
challenge	5
credentials	6
G	
Grammar	
Authorization	8
Proxy-Authenticate	9
Proxy-Authorization	9
WWW-Authenticate	9
H	
Header Fields	
Authorization	8
Proxy-Authenticate	9
Proxy-Authorization	9
WWW-Authenticate	9
P	
Proxy-Authenticate header field	9
Proxy-Authorization header field	9
R	
realm	5
realm-value	5

S

Status Codes

401 Unauthorized 7

407 Proxy Authentication Required 7

W

WWW-Authenticate header field 9

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/>

Jim Gettys
Alcatel-Lucent Bell Labs
21 Oak Knoll Road
Carlisle, MA 01741
USA

EMail: jg@freedesktop.org
URI: <http://gettys.wordpress.com/>

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

EMail: JeffMogul@acm.org

Henrik Frystyk Nielsen
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052
USA

EMail: henrikn@microsoft.com

Larry Masinter
Adobe Systems Incorporated
345 Park Ave
San Jose, CA 95110
USA

EMail: LMM@acm.org
URI: <http://larry.masinter.net/>

Paul J. Leach
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052

EMail: paulle@microsoft.com

Tim Berners-Lee
World Wide Web Consortium
MIT Computer Science and Artificial Intelligence Laboratory
The Stata Center, Building 32
32 Vassar Street
Cambridge, MA 02139
USA

EMail: timbl@w3.org
URI: <http://www.w3.org/People/Berners-Lee/>

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

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

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

Phone: +49 251 2807760
Fax: +49 251 2807761
EMail: julian.reschke@greenbytes.de
URI: <http://greenbytes.de/tech/webdav/>

