HTTP Working Group
INTERNET-DRAFT
<draft-ietf-http-v11-spec-rev-05>

R. Fielding, UC Irvine
J. Gettys, Compaq/W3C
J. C. Mogul, Compaq
H. Frystyk, W3C/MIT
L. Masinter, Xerox
P. Leach, Microsoft
T. Berners-Lee, W3C/MIT

Expires March 11, 1999
September 11, 1998

Hypertext Transfer Protocol -- HTTP/1.1

Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

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

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

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-9 [46]. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers [47]. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred.

HTTP has been in use by the World-Wide Web global information initiative since 1990. This specification defines the protocol referred to as "HTTP/1.1", and is an update to RFC 2068 [33].

Copyright Notice

Table of Contents

HYPERTEXT TRANSFER PROTOCOL -- HTTP/1.1 1

Status of this Memo 1

Abstract 1

Copyright Notice 1

Table of Contents 3

1 Introduction 8
1.1 Purpose 8
1.2 Requirements 8
1.3 Terminology 9
1.4 Overall Operation 11

2 Notational Conventions and Generic Grammar 12
2.1 Augmented BNF 12
2.2 Basic Rules 13

3 Protocol Parameters 14
3.1 HTTP Version 14
3.2 Uniform Resource Identifiers 15
3.2.1 General Syntax 15
3.2.2 http URL 15
3.2.3 URI Comparison 16
3.3 Date/Time Formats 16
3.3.1 Full Date 16
3.3.2 Delta Seconds 17
3.4 Character Sets 17
3.4.1 Missing Charset 17
3.5 Content Codings 18
3.6 Transfer Codings 18
3.6.1 Chunked Transfer Coding 19
3.7 Media Types 20
3.7.1 Canonicalization and Text Defaults 20
3.7.2 Multipart Types 20
3.8 Product Tokens 21
3.9 Quality Values 21
3.10 Language Tags 21
3.11 Entity Tags 22
3.12 Range Units 22

4 HTTP Message 22
4.1 Message Types 22
4.2 Message Headers 23
4.3 Message Body 23
4.4 Message Length 24
4.5 General Header Fields 24

5 Request 25
5.1 Request-Line 25
5.1.1 Method 25
5.1.2 Request-URI 25
5.2 The Resource Identified by a Request 26
5.3 Request Header Fields 27

6 Response 27
6.1 Status-Line 27
6.1.1 Status Code and Reason Phrase 27
6.2 Response Header Fields 29

7 Entity 29
7.1 Entity Header Fields 29
7.2 Entity Body 30
7.2.1 Type 30
7.2.2 Entity Length 30

8 Connections 30
8.1 Persistent Connections 30
8.1.1 Purpose 30
8.1.2 Overall Operation 31
8.1.3 Proxy Servers 32
8.1.4 Practical Considerations 32
8.2 Message Transmission Requirements 32
8.2.1 Persistent Connections and Flow Control 32
8.2.2 Monitoring Connections for Error Status Messages 33
8.2.3 Automatic Retrying of Requests 33
8.2.4 Use of the 100 (Continue) Status 33
8.2.5 Client Behavior if Server Prematurely Closes Connection 34

9 Method Definitions 34
9.1 Safe and Idempotent Methods 35
9.1.1 Safe Methods 35
9.1.2 Idempotent Methods 35
9.2 OPTIONS 35
9.3 GET 36
9.4 HEAD 36
9.5 POST 36
9.6 PUT 37
9.7 DELETE 38
9.8 TRACE 38
9.9 CONNECT 38

10 Status Code Definitions 38
10.1 Informational 1xx 38
10.1.1 100 Continue 39
10.1.2 101 Switching Protocols 39
10.2 Successful 2xx 39
10.2.1 200 OK 39
10.2.2 201 Created 39
10.2.3 202 Accepted 39
10.2.4 203 Non-Authoritative Information 40
10.2.5 204 No Content 40
10.2.6 205 Reset Content 40

10.2.7 206 Partial Content 40
10.3 Redirection 3xx 41
10.3.1 300 Multiple Choices 41
10.3.2 301 Moved Permanently 41
10.3.3 302 Found 41
10.3.4 303 See Other 42
10.3.5 304 Not Modified 42
10.3.6 305 Use Proxy 43
10.3.7 306 (Unused) 43
10.3.8 307 Temporary Redirect 43
10.4 Client Error 4xx 43
10.4.1 400 Bad Request 43
10.4.2 401 Unauthorized 43
10.4.3 402 Payment Required 44
10.4.4 403 Forbidden 44
10.4.5 404 Not Found 44
10.4.6 405 Method Not Allowed 44
10.4.7 406 Not Acceptable 44
10.4.8 407 Proxy Authentication Required 44
10.4.9 408 Request Timeout 45
10.4.10 409 Conflict 45
10.4.11 410 Gone 45
10.4.12 411 Length Required 45
10.4.13 412 Precondition Failed 45
10.4.14 413 Request Entity Too Large 45
10.4.15 414 Request-URI Too Long 45
10.4.16 415 Unsupported Media Type 46
10.4.17 416 Requested Range Not Satisfiable 46
10.4.18 417 Expectation Failed 46
10.5 Server Error 5xx 46
10.5.1 500 Internal Server Error 46
10.5.2 501 Not Implemented 46
10.5.3 502 Bad Gateway 46
10.5.4 503 Service Unavailable 46
10.5.5 504 Gateway Timeout 47
10.5.6 505 HTTP Version Not Supported 47

11 Access Authentication 47

12 Content Negotiation 47
12.1 Server-driven Negotiation 47
12.2 Agent-driven Negotiation 48
12.3 Transparent Negotiation 48

13 Caching in HTTP 49
13.1.1 Cache Correctness 49
13.1.2 Warnings 50
13.1.3 Cache-control Mechanisms 51
13.1.4 Explicit User Agent Warnings 51
13.1.5 Exceptions to the Rules and Warnings 51
13.1.6 Client-controlled Behavior 51
13.2 Expiration Model 52
13.2.1 Server-Specified Expiration 52
13.2.2 Heuristic Expiration 52
13.2.3 Age Calculations 52

13.2.4 Expiration Calculations 54
13.2.5 Disambiguating Expiration Values 54
13.2.6 Disambiguating Multiple Responses 54
13.3 Validation Model 55
13.3.1 Last-modified Dates 55
13.3.2 Entity Tag Cache Validators 55
13.3.3 Weak and Strong Validators 56
13.3.4 Rules for When to Use Entity Tags and Last-modified Dates 57
13.3.5 Non-validating Conditionals 58
13.4 Response Cachability 58
13.5 Constructing Responses From Caches 59
13.5.1 End-to-end and Hop-by-hop Headers 59
13.5.2 Non-modifiable Headers 59
13.5.3 Combining Headers 60
13.5.4 Combining Byte Ranges 61
13.6 Caching Negotiated Responses 61
13.7 Shared and Non-Shared Caches 62
13.8 Errors or Incomplete Response Cache Behavior 62
13.9 Side Effects of GET and HEAD 62
13.10 Invalidation After Updates or Deletions 62
13.11 Write-Through Mandatory 63
13.12 Cache Replacement 63
13.13 History Lists 63

14 Header Field Definitions 63
14.1 Accept 63
14.2 Accept-Charset 65
14.3 Accept-Encoding 65
14.4 Accept-Language 66
14.5 Accept-Ranges 67
14.6 Age 67
14.7 Allow 67
14.8 Authorization 67
14.9 Cache-Control 68
14.9.1 What is Cachable 69
14.9.2 What May be Stored by Caches 70
14.9.3 Modifications of the Basic Expiration Mechanism 70
14.9.4 Cache Revalidation and Reload Controls 71
14.9.5 No-Transform Directive 73
14.9.6 Cache Control Extensions 73
14.10 Connection 73
14.11 Content-Encoding 74
14.12 Content-Language 74
14.13 Content-Length 75
14.14 Content-Location 75
14.15 Content-MD5 76
14.16 Content-Range 76
14.17 Content-Type 78
14.18 Date 78
14.18.1 Clockless Origin Server Operation 78
14.19 ETag 79
14.20 Expect 79
14.20.1 Expect 100-continue 79
14.21 Expires 80
14.22 From 80

14.23 Host 81
14.24 If-Match 81
14.25 If-Modified-Since 82
14.26 If-None-Match 82
14.27 If-Range 83
14.28 If-Unmodified-Since 84
14.29 Last-Modified 84
14.30 Location 84
14.31 Max-Forwards 85
14.32 Pragma 85
14.33 Proxy-Authenticate 85
14.34 Proxy-Authorization 86
14.35 Range 86
14.35.1 Byte Ranges 86
14.35.2 Range Retrieval Requests 87
14.36 Referer 87
14.37 Retry-After 88
14.38 Server 88
14.39 TE 88
14.40 Trailer 89
14.41 Transfer-Encoding 89
14.42 Upgrade 90
14.43 User-Agent 90
14.44 Vary 90
14.45 Via 91
14.46 Warning 92
14.47 WWW-Authenticate 93

15 Security Considerations 94
15.1 Personal Information 94
15.1.1 Abuse of Server Log Information 94
15.1.2 Transfer of Sensitive Information 94
15.1.3 Encoding Sensitive Information in URI's 95
15.1.4 Privacy Issues Connected to Accept Headers 95
15.2 Attacks Based On File and Path Names 95
15.3 DNS Spoofing 96
15.4 Location Headers and Spoofing 96
15.5 Content-Disposition Issues 96
15.6 Authentication Credentials and Idle Clients 96
15.7 Proxies and Caching 97
15.7.1 Denial of Service Attacks on Proxies 97

16 Acknowledgments 97

17 References 98

18 Authors' Addresses 101

19 Appendices 102
19.1 Internet Media Type message/http and application/http 102
19.2 Internet Media Type multipart/byteranges 102
19.3 Tolerant Applications 103
19.4 Differences Between HTTP Entities and RFC 2045 Entities 104
19.4.1 MIME-Version 104
19.4.2 Conversion to Canonical Form 104

19.4.3 Conversion of Date Formats 104
19.4.4 Introduction of Content-Encoding 105
19.4.5 No Content-Transfer-Encoding....105
19.4.6 Introduction of Transfer-Encoding 105
19.4.7 MHTML and Line Length Limitations 105
19.5 Additional Features....105
19.5.1 Content-Disposition 106
19.6 Compatibility with Previous Versions 106
19.6.1 Changes from HTTP/1.0....106
19.6.2 Compatibility with HTTP/1.0 Persistent Connections 107
19.6.3 Changes from RFC 2068....107
19.7 Notes to the RFC Editor and IANA 109
19.7.1 Transfer-coding Values....109
19.7.2 Definition of application/http....109
19.7.3 Addition of "`identity`" content-coding to content-coding Registry....109

20 Full Copyright Statement....109

21 Index 110

1 Introduction

1.1 Purpose

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. The first version of HTTP, referred to as HTTP/0.9, was a simple protocol for raw data transfer across the Internet. HTTP/1.0, as defined by RFC 1945 [6], improved the protocol by allowing messages to be in the format of MIME-like messages, containing metainformation about the data transferred and modifiers on the request/response semantics. However, HTTP/1.0 does not sufficiently take into consideration the effects of hierarchical proxies, caching, the need for persistent connections, or virtual hosts. In addition, the proliferation of incompletely-implemented applications calling themselves "HTTP/1.0" has necessitated a protocol version change in order for two communicating applications to determine each other's true capabilities.

This specification defines the protocol referred to as "HTTP/1.1". This protocol includes more stringent requirements than HTTP/1.0 in order to ensure reliable implementation of its features.

Practical information systems require more functionality than simple retrieval, including search, front-end update, and annotation. HTTP allows an open-ended set of methods and headers that indicate the purpose of a request [47]. It builds on the discipline of reference provided by the Uniform Resource Identifier (URI) [3], as a location (URL) [4] or name (URN) [20], for indicating the resource to which a method is to be applied. Messages are passed in a format similar to that used by Internet mail [9] as defined by the Multipurpose Internet Mail Extensions (MIME) [7].

HTTP is also used as a generic protocol for communication between user agents and proxies/gateways to other Internet systems, including those supported by the SMTP [16], NNTP [13], FTP [18], Gopher [2], and WAIS [10] protocols. In this way, HTTP allows basic hypermedia access to resources available from diverse applications.

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 RFC 2119 [34].

An implementation is not compliant if it fails to satisfy one or more of the MUST level requirements for the protocols it implements. An implementation that satisfies all the MUST 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 Terminology

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

connection
: A transport layer virtual circuit established between two programs for the purpose of communication.

message
: The basic unit of HTTP communication, consisting of a structured sequence of octets matching the syntax defined in section 4 and transmitted via the connection.

request
: An HTTP request message, as defined in section 5.

response
: An HTTP response message, as defined in section 6.

resource
: A network data object or service that can be identified by a URI, as defined in section 3.2. Resources may be available in multiple representations (e.g. multiple languages, data formats, size, and resolutions) or vary in other ways.

entity
: The information transferred as the payload of a request or response. An entity consists of metainformation in the form of entity-header fields and content in the form of an entity-body, as described in section 7.

representation
: An entity included with a response that is subject to content negotiation, as described in section 12. There may exist multiple representations associated with a particular response status.

content negotiation
: The mechanism for selecting the appropriate representation when servicing a request, as described in section 12. The representation of entities in any response can be negotiated (including error responses).

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

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

user agent
: The client which initiates a request. These are often browsers, editors, spiders (web-traversing robots), or other end user tools.

server
: An application program that accepts connections in order to service requests by sending back responses. Any given program may be capable of being both a client and a server; our use of these terms refers only to the role being performed by the program for a particular connection, rather than to the program’s capabilities in general. Likewise, any server may act as an origin server, proxy, gateway, or tunnel, switching behavior based on the nature of each request.

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

proxy
: An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, with possible translation, to other servers. A proxy MUST implement both the client and server requirements of this specification. A “transparent proxy” is a proxy that does not modify the request or response beyond what is required for proxy authentication and identification. A “non-transparent proxy” is a proxy that modifies the request or response in order to provide some added service to the user agent, such as group annotation services, media type transformation, protocol reduction, or anonymity filtering. Except where either transparent or non-transparent behavior is explicitly stated, the HTTP proxy requirements apply to both types of proxies.

gateway
: A server which acts as an intermediary for some other server. Unlike a proxy, a gateway receives requests as if it were the origin server for the requested resource; the requesting client may not be aware that it is communicating with a gateway.

tunnel
: An intermediary program which is acting as a blind relay between two connections. Once active, a tunnel is not considered a party to the HTTP communication, though the tunnel may have been initiated by an HTTP request. The tunnel ceases to exist when both ends of the relayed connections are closed.

cache
: A program’s local store of response messages and the subsystem that controls its message storage, retrieval, and deletion. A cache stores cachable responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests. Any client or server may include a cache, though a cache cannot be used by a server that is acting as a tunnel.

cachable
: A response is cachable if a cache is allowed to store a copy of the response message for use in answering subsequent requests. The rules for determining the cachability of HTTP responses are defined in section 13. Even if a resource is cachable, there may be additional constraints on whether a cache can use the cached copy for a particular request.

first-hand
: A response is first-hand if it comes directly and without unnecessary delay from the origin server, perhaps via one or more proxies. A response is also first-hand if its validity has just been checked directly with the origin server.

explicit expiration time
: The time at which the origin server intends that an entity should 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.

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.

semantically transparent
: A cache behaves in a "semantically transparent" manner, with respect to a particular response, when its use affects neither the requesting client nor the origin server, except to improve performance. When a cache is semantically transparent, the client receives exactly the same response (except for hop-by-hop headers) that it would have received had its request been handled directly by the origin server.

validator
: A protocol element (e.g., an entity tag or a Last-Modified time) that is used to find out whether a cache entry is an equivalent copy of an entity.

upstream/downstream
: Upstream and downstream describe the flow of a message: all messages flow from upstream to downstream.

inbound/outbound
: Inbound and outbound refer to the request and response paths for messages: "inbound" means "travelling toward the origin server", and "outbound" means "travelling toward the user agent"

1.4 Overall Operation

The HTTP protocol is a request/response protocol. A client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a connection with a server. The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity metainformation, and possible entity-body content. The relationship between HTTP and MIME is described in appendix 19.4.

Most HTTP communication is initiated by a user agent and consists of a request to be applied to a resource on some origin server. In the simplest case, this may be accomplished via a single connection (v) between the user agent (UA) and the origin server (O).

```
       request chain ------------------------>
    UA -------------------v------------------- O
       <----------------------- response chain
```

A more complicated situation occurs when one or more intermediaries are present in the request/response chain. There are three common forms of intermediary: proxy, gateway, and tunnel. A proxy is a forwarding agent, receiving requests for a URI in its absolute form, rewriting all or part of the message, and forwarding the reformatted request toward the server identified by the URI. A gateway is a receiving agent, acting as a layer above some other server(s) and, if necessary, translating the requests to the underlying server's protocol. A tunnel acts as a relay point between two connections without changing the messages; tunnels are used when the communication needs to pass through an intermediary (such as a firewall) even when the intermediary cannot understand the contents of the messages.

```
       request chain -------------------------------------->
    UA -----v----- A -----v----- B -----v----- C -----v----- O
       <------------------------------------- response chain
```

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. This distinction is important because some HTTP communication options may 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 may be engaged in multiple, simultaneous communications. For example, B may 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.

Any party to the communication which is not acting as a tunnel may employ an internal cache for handling requests. 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.

```
          request chain ---------->
       UA -----v----- A -----v----- B - - - - - - C - - - - - - O
          <--------- response chain
```

Not all responses are usefully cachable, and some requests may contain modifiers which place special requirements on cache behavior. HTTP requirements for cache behavior and cachable responses are defined in section 13.

In fact, there are a wide variety of architectures and configurations of caches and proxies currently being experimented with or deployed across the World Wide Web. 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 CD-ROM, and so on. HTTP systems are used in corporate intranets over high-bandwidth links, and for access via PDAs with low-power radio links and intermittent connectivity. The goal of HTTP/1.1 is to support the wide diversity of configurations already deployed while introducing protocol constructs that meet the needs of those who build web applications that require high reliability and, failing that, at least reliable indications of failure.

HTTP communication usually takes place over TCP/IP connections. The default port is TCP 80 [19], but other ports can be used. This does not preclude HTTP from being implemented on top of any other protocol on the Internet, or on other networks. HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used; the mapping of the HTTP/1.1 request and response structures onto the transport data units of the protocol in question is outside the scope of this specification.

In HTTP/1.0, most implementations used a new connection for each request/response exchange. In HTTP/1.1, a connection may be used for one or more request/response exchanges, although connections may be closed for a variety of reasons (see section 8.1).

2 Notational Conventions and Generic Grammar

2.1 Augmented BNF

All of the mechanisms specified in this document are described in both prose and an augmented Backus-Naur Form (BNF) similar to that used by RFC 822 [9]. Implementors will need to be familiar with the notation in order to understand this specification. The augmented BNF includes the following constructs:

`name = definition`

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

`"literal"`

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

`rule1 | rule2`

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

`(rule1 rule2)`

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

`*rule`

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

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

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

#rule
A construct “#” is defined, similar to “*”, for defining lists of elements. The full form is “<n>#<m>element” indicating at least <n> and at most <m> elements, each separated by one or more commas (",") and OPTIONAL linear white space (LWS). This makes the usual form of lists very easy; a rule such as

```
( *LWS element *( *LWS "," *LWS element ))
```

can be shown as

```
1#element
```

Wherever this construct is used, null elements are allowed, but do not contribute to the count of elements present. That is, “(element), , (element) ” is permitted, but counts as only two elements. Therefore, where at least one element is required, at least one non-null element MUST be present. Default values are 0 and infinity so that “#element” allows any number, including zero; “1#element” requires at least one; and “1#2element” allows one or two.

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

implied *LWS
The grammar described by this specification is word-based. Except where noted otherwise, linear white space (LWS) can be included between any two adjacent words (token or quoted-string), and between adjacent tokens and separators, without changing the interpretation of a field. At least one delimiter (LWS and/or separators[jg13]) MUST exist between any two tokens (for the definition of “token” below), since they would otherwise be interpreted as a single token.

2.2 Basic Rules

The following rules are used throughout this specification to describe basic parsing constructs. The US-ASCII coded character set is defined by ANSI X3.4-1986 [21].

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

HTTP/1.1 defines the sequence CR LF as the end-of-line marker for all protocol elements except the entity-body (see appendix 19.3 for tolerant applications). The end-of-line marker within an entity-body is defined by its associated media type, as described in section 3.7.

```
       CRLF           = CR LF
```

HTTP/1.1 header field values can be folded onto multiple lines if

ERROR: syntaxerror
OFFENDING COMMAND: ----nostringval----

STACK:

0
-0.074