Wikipedia

HTTP

The **Hypertext Transfer Protocol** (**HTTP**) is an application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems.[1] HTTP is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlinks to other resources that the user can easily access, for example by a mouse click or by tapping the screen in a web browser.

Development of HTTP was initiated by Tim Berners-Lee at CERN in 1989 and summarized in a simple document describing the behavior of a client and a server using the first HTTP version, named 0.9.[2] That version was subsequently developed, eventually becoming the public 1.0.[3]

Development of early HTTP Requests for Comments (RFCs) started a few years later in a coordinated effort by the Internet Engineering Task Force (IETF) and the

World Wide Web Consortium (W3C), with work later moving to the IETF.

HTTP/1 was finalized and fully documented (as version 1.0) in 1996.[4] It evolved (as version 1.1) in 1997 and then its specifications were updated in 1999, 2014, and 2022.[5]

HTTP

International standard	RFC 19 s://da r.ietf. html/ 5) H RFC 91 s://da r.ietf. html/ 0) H Sema RFC 91 s://da

Its secure variant named HTTPS is used by more than 80% of websites.[6] HTTP/2, published in 2015, provides a more efficient expression of HTTP's semantics "on the wire". As of April 2023, it is used by 39% of websites[7] and supported by almost all web browsers (over 97%

r.ietf.
html/
1). H
Cach
RFC 91
s://da
r.ietf.
html/
2). H
RFC 91
s://da
r.ietf.
html/
3). H
RFC 75
s://da
r.ietf.

of users).[8] It is also supported by major web servers over Transport Layer Security (TLS) using an Application-Layer Protocol Negotiation (ALPN) extension[9] where TLS 1.2 or newer is required.[10][11]

HTTP/3, the successor to HTTP/2, was published in

html/
1). H
HPAC
Heac
Comp
RFC 81
s://da
r.ietf.
html/
4). H
Oppo
Secu
HTTP
RFC 83
s://da
r.ietf.
html/

2022.[12] It is now used on over 26% of websites[13] and is supported by most web browsers, i.e. (at least partially) supported by 94% of web browsers.[14] HTTP/3 uses QUIC instead of TCP for the underlying transport protocol. Like HTTP/2, it does not obsolesce previous major versions of the

6). H
The (
HTTF
Fram
RFC 84
s://da
r.ietf.
html/
1). H
Boot
WebS
with
RFC 91
s://da
r.ietf.
html/
4). H

	RFC 92 s://da r.ietf. html/ 4). H QPAC Com
Developed by	initially CERN; IETF, W3C
Introduced	1991
Website	https://httpv

protocol. Support for HTTP/3 was added to Cloudflare and Google Chrome first,[15][16] and is also enabled in Firefox.[17] HTTP/3 has lower latency for real-world web pages, if enabled on the server, load faster than with HTTP/2, and even faster than HTTP/1.1, in some cases over 3× faster than HTTP/1.1 (which is still commonly only enabled).[18]

Technical overview

URL beginning with the HTTP scheme and the WWW domain name label

HTTP functions as a request–response protocol in the client–server model. A web browser, for example, may be the *client* whereas a process, named web server, running on a computer hosting one or more websites may be the *server*. The client submits an HTTP *request* message to the server. The server, which provides *resources* such as HTML files and other content or performs other functions on behalf of the client, returns a *response* message to the client. The response contains completion status information about the request and may also contain requested content in its message body.

A web browser is an example of a *user agent* (UA). Other types of user agent include the indexing software used by search providers (web crawlers), voice browsers, mobile apps, and other software that accesses, consumes, or displays web content.

HTTP is designed to permit intermediate network elements to improve or enable communications between clients and servers. High-traffic websites often benefit from web cache servers that deliver content on behalf of upstream servers to improve response time. Web browsers cache previously accessed web resources and reuse them, whenever possible, to reduce network traffic. HTTP proxy servers at private network boundaries can facilitate communication for clients without a globally routable address, by relaying messages with external servers.

To allow intermediate HTTP nodes (proxy servers, web caches, etc.) to accomplish their functions, some of the HTTP headers (found in HTTP requests/responses) are managed hop-by-hop whereas other HTTP headers are managed end-to-end (managed only by the source client and by the target web server).

HTTP is an application layer protocol designed within the framework of the Internet protocol suite. Its definition presumes an underlying and reliable transport layer protocol.[19] In the latest version HTTP/3, the Transmission Control Protocol (TCP) is no longer used, but the older versions are still more used and they most commonly use TCP. They have also been adapted to use unreliable protocols such as the User Datagram Protocol (UDP), which HTTP/3 also (indirectly) always builds on, for example in HTTPU and Simple Service Discovery Protocol (SSDP).

HTTP resources are identified and located on the network by Uniform Resource Locators (URLs), using the Uniform Resource Identifiers (URI's) schemes *http* and *https*. As defined in RFC 3986 (https://datatracker.ietf.org/doc/html/rfc3986) , URIs are encoded as hyperlinks in HTML documents, so as to form interlinked hypertext documents.

In HTTP/1.0 a separate TCP connection to the same server is made for every resource request.[20]

In HTTP/1.1 instead a TCP connection can be reused to make multiple resource requests (i.e. of HTML pages, frames, images, scripts, stylesheets, etc.).[21][22]

HTTP/1.1 communications therefore experience less latency as the establishment of TCP connections presents considerable overhead, specially under high traffic conditions.[23]

HTTP/2 is a revision of previous HTTP/1.1 in order to maintain the same client–server model and the same protocol methods but with these differences in order:

- to use a compressed binary representation of metadata (HTTP headers) instead of a textual one, so that headers require much less space;
- to use a single TCP/IP (usually encrypted) connection per accessed server domain instead of 2 to 8 TCP/IP connections;
- to use one or more bidirectional streams per TCP/IP connection in which HTTP

requests and responses are broken down and transmitted in small packets to almost solve the problem of the HOLB (head-of-line blocking).[note 1]

- to add a push capability to allow server application to send data to clients whenever new data is available (without forcing clients to request periodically new data to server by using polling methods).[24]

HTTP/2 communications therefore experience much less latency and, in most cases, even more speed than HTTP/1.1 communications.

HTTP/3 is a revision of previous HTTP/2 in order to use QUIC + UDP transport protocols instead of TCP. Before that version TCP/IP connections were used, now only the IP layer (which UDP, like TCP, builds on) also to slightly improve the average speed of communications and to avoid the occasional (very rare) problem of TCP connection congestion that can temporarily block or slow down the data flow of all its streams (another form of "*head of line blocking*").

History

Tim Berners-Lee

The term hypertext was coined by Ted Nelson in 1965 in the Xanadu Project, which was in turn inspired by Vannevar Bush's 1930s vision of the microfilm-based information retrieval and management "memex" system described in his 1945 essay "As We May Think". Tim Berners-Lee and his team at CERN are credited with inventing the original HTTP, along with HTML and the associated technology for a web server and a client user interface called web browser. Berners-Lee designed HTTP in order to help with the adoption of his other idea: the "WorldWideWeb" project, which was first proposed in 1989, now known as the World Wide Web.

The first web server went live in 1990.[25][26] The protocol used had only one method, namely GET, which would request a page from a server.[27] The response from the server was always an HTML page.[2]

Summary of HTTP milestone versions

Version	Year introduced	Current status
HTTP/0.9	1991	Obsolete
HTTP/1.0	1996	Obsolete
HTTP/1.1	1997	Standard
HTTP/2	2015	Standard
HTTP/3	2022	Standard

HTTP/0.9

In 1991, the first documented official version of HTTP was written as a plain document, less than 700 words long, and this version was named HTTP/0.9, which supported only GET method, allowing clients to only retrieve HTML documents from the server, but not supporting any other file formats or information upload.[2]

HTTP/1.0-draft

Since 1992, a new document was written to specify the evolution of the basic protocol towards its next full version. It supported both the simple request method of the 0.9 version and the full GET request that included the client HTTP version. This was the first of the many unofficial HTTP/1.0 drafts that preceded the final work on HTTP/1.0.[3]

W3C HTTP Working Group

After having decided that new features of HTTP protocol were required and that they had to be fully documented as official RFCs, in early 1995 the HTTP Working Group (HTTP WG, led by Dave

Raggett) was constituted with the aim to standardize and expand the protocol with extended operations, extended negotiation, richer meta-information, tied with a security protocol which became more efficient by adding additional methods and header fields.[28][29]

The HTTP WG planned to revise and publish new versions of the protocol as HTTP/1.0 and HTTP/1.1 within 1995, but, because of the many revisions, that timeline lasted much more than one year.[30]

The HTTP WG planned also to specify a far future version of HTTP called HTTP-

NG (HTTP Next Generation) that would have solved all remaining problems, of previous versions, related to performances, low latency responses, etc. but this work started only a few years later and it was never completed.

HTTP/1.0

In May 1996, RFC 1945 (https://datatracker.ietf.org/doc/html/rfc1945) was published as a final HTTP/1.0 revision of what had been used in previous 4 years as a pre-standard HTTP/1.0-draft which was already used by many web browsers and web servers.

In early 1996 developers started to even include unofficial extensions of the

HTTP/1.0 protocol (i.e. keep-alive connections, etc.) into their products by using drafts of the upcoming HTTP/1.1 specifications.[31]

HTTP/1.1

Since early 1996, major web browsers and web server developers also started to implement new features specified by pre-standard HTTP/1.1 drafts specifications. End-user adoption of the new versions of browsers and servers was rapid. In March 1996, one web hosting company reported that over 40% of browsers in use on the Internet used the new HTTP/1.1 header "Host" to enable virtual hosting. That same web hosting company reported that by June 1996, 65% of all browsers accessing their servers were pre-standard HTTP/1.1 compliant.[32]

In January 1997, RFC 2068 (https://datatracker.ietf.org/doc/html/rfc2068) was officially released as HTTP/1.1 specifications.

In June 1999, RFC 2616 (https://datatracker.ietf.org/doc/html/rfc2616) was released to include all improvements and updates based on previous (obsolete) HTTP/1.1 specifications.

W3C HTTP-NG Working Group

Resuming the old 1995 plan of previous HTTP Working Group, in 1997 an *HTTP-*

NG Working Group was formed to develop a new HTTP protocol named HTTP-NG (HTTP New Generation). A few proposals / drafts were produced for the new protocol to use multiplexing of HTTP transactions inside a single TCP/IP connection, but in 1999, the group stopped its activity passing the technical problems to IETF.[33]

IETF HTTP Working Group restarted

In 2007, the IETF HTTP Working Group (https://httpwg.org/) (HTTP WG bis or HTTPbis) was restarted firstly to revise and clarify previous HTTP/1.1 specifications and secondly to write and refine future HTTP/2 specifications (named httpbis).[34][35]

SPDY: an unofficial HTTP protocol developed by Google

In 2009, Google, a private company, announced that it had developed and tested a new HTTP binary protocol named SPDY. The implicit aim was to greatly speed up web traffic (specially between future web browsers and its servers).

SPDY was indeed much faster than HTTP/1.1 in many tests and so it was quickly adopted by Chromium and then by other major web browsers.[36]

Some of the ideas about multiplexing HTTP streams over a single TCP/IP connection were taken from various sources, including the work of W3C HTTP-NG Working Group.

HTTP/2

In January–March 2012, HTTP Working Group (HTTPbis) announced the need to start to focus on a new HTTP/2 protocol (while finishing the revision of HTTP/1.1 specifications), maybe taking in consideration ideas and work done for SPDY.[37][38]

After a few months about what to do to develop a new version of HTTP, it was decided to derive it from SPDY.[39]

In May 2015, HTTP/2 was published as RFC 7540 (https://datatracker.ietf.org/doc/html/rfc7540) and quickly adopted by all web browsers already supporting SPDY and more slowly by web servers.

2014 updates to HTTP/1.1

In June 2014, the HTTP Working Group released an updated six-part HTTP/1.1 specification obsoleting RFC 2616 (https://datatracker.ietf.org/doc/html/rfc2616) :

- RFC 7230 (https://datatracker.ietf.org/doc/html/rfc7230) , *HTTP/1.1: Message Syntax and Routing*

- RFC 7231 (https://datatracker.ietf.org/doc/html/rfc7231) , *HTTP/1.1: Semantics and Content*
- RFC 7232 (https://datatracker.ietf.org/doc/html/rfc7232) , *HTTP/1.1: Conditional Requests*
- RFC 7233 (https://datatracker.ietf.org/doc/html/rfc7233) , *HTTP/1.1: Range Requests*
- RFC 7234 (https://datatracker.ietf.org/doc/html/rfc7234) , *HTTP/1.1: Caching*
- RFC 7235 (https://datatracker.ietf.org/doc/html/rfc7235) , *HTTP/1.1: Authentication*

HTTP/0.9 Deprecation

In RFC 7230 (https://datatracker.ietf.org/doc/html/rfc7230). Appendix-A, HTTP/0.9 was deprecated for servers supporting HTTP/1.1 version (and higher):[40]

> *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 caused by a client failing to properly encode the request-target.*

Since 2016 many product managers and developers of user agents (browsers, etc.) and web servers have begun planning to gradually deprecate and dismiss support for HTTP/0.9 protocol, mainly for the following reasons:[41]

- it is so simple that an RFC document was never written (there is only the original document);[2]
- it has no HTTP headers and lacks many other features that nowadays are required for minimal security reasons;
- it has not been widespread since 1999..2000 (because of HTTP/1.0 and HTTP/1.1) and is commonly used only by some very old network hardware, i.e. routers, etc.

[note 2]

HTTP/3

In 2020, the first drafts HTTP/3 were published and major web browsers and web servers started to adopt it.
On 6 June 2022, IETF standardized HTTP/3 as RFC 9114 (https://datatracker.ietf.org/doc/html/rfc9114) .[42]

Updates and refactoring in 2022

In June 2022, a batch of RFCs was published, deprecating many of the previous documents and introducing a few minor changes and a refactoring of HTTP semantics description into a separate document.

- RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110) , *HTTP Semantics*

- RFC 9111 (https://datatracker.ietf.org/doc/html/rfc9111) , *HTTP Caching*
- RFC 9112 (https://datatracker.ietf.org/doc/html/rfc9112) , *HTTP/1.1*
- RFC 9113 (https://datatracker.ietf.org/doc/html/rfc9113) , *HTTP/2*
- RFC 9114 (https://datatracker.ietf.org/doc/html/rfc9114) , *HTTP/3* (see also the section above)
- RFC 9204 (https://datatracker.ietf.org/doc/html/rfc9204) , *QPACK: Field Compression for HTTP/3*
- RFC 9218 (https://datatracker.ietf.org/doc/html/rfc9218) , *Extensible Prioritization Scheme for HTTP*

HTTP data exchange

HTTP is a stateless application-level protocol and it requires a reliable network transport connection to exchange data between client and server.[19] In HTTP implementations, TCP/IP connections are used using well known ports (typically port 80 if the connection is unencrypted or port 443 if the connection is encrypted, see also List of TCP and UDP port numbers).[43][44] In HTTP/2, a TCP/IP connection plus multiple protocol channels are used. In HTTP/3, the application transport protocol QUIC over UDP is used.

Request and response messages through connections

Data is exchanged through a sequence of request–response messages which are exchanged by a session layer transport connection.[19] An HTTP client initially tries to connect to a server establishing a connection (real or virtual). An HTTP(S) server listening on that port accepts the connection and then waits for a client's request message. The client sends its request to the server. Upon receiving the request, the server sends back an HTTP response message (header plus a body if it is required). The body of this message is typically the requested resource, although an error message or other information may also be returned. At any time (for many reasons) client or server can close the connection. Closing a connection is usually advertised in advance by using one or more HTTP headers in the last request/response message sent to server or client.[21]

Persistent connections

In **HTTP/0.9**, the TCP/IP connection is always closed after server response has been sent, so it is never persistent.

In **HTTP/1.0**, as stated in RFC 1945, the TCP/IP connection should always be closed by server after a response has been sent.[note 3]

In **HTTP/1.1** a keep-alive-mechanism was officially introduced so that a connection could be reused for more than one request/response. Such persistent connections reduce request latency perceptibly because the client does not need to re-negotiate the TCP 3-Way-Handshake connection after the first request has been sent. Another positive side effect is that, in general, the connection becomes faster with time due to TCP's slow-start-mechanism.

HTTP/1.1 added also HTTP pipelining in order to further reduce lag time when using persistent connections by allowing clients to send multiple requests before waiting for each response. This optimization was never considered really safe because a few web servers and many proxy servers, specially transparent proxy servers placed in Internet / Intranets between clients and servers, did not handle pipelined requests properly (they served only the first request discarding the others, they closed the connection because they saw more data after the first request or some proxies even returned responses out of order etc.). Besides this only HEAD and some GET requests (i.e. limited to real file requests and so with URLs without query string used as a command, etc.) could be pipelined in a safe and idempotent mode. After many years of struggling with the problems introduced by enabling pipelining, this feature was first disabled and then removed from most browsers also because of the announced adoption of HTTP/2.

HTTP/2 extended the usage of persistent connections by multiplexing many concurrent requests/responses through a single TCP/IP connection.

HTTP/3 does not use TCP/IP connections but QUIC + UDP (see also: technical overview).

Content retrieval optimizations

HTTP/0.9
a requested resource was always sent entirely.

HTTP/1.0

HTTP/1.0 added headers to manage resources cached by client in order to allow conditional GET requests; in practice a server has to return the entire content of the requested resource only if its last modified time is not known by client or if it changed since last full response to GET request. One of these headers, "Content-Encoding", was added to specify whether the returned content of a resource was or was not compressed.

If the total length of the content of a resource was not known in advance (i.e. because it was dynamically generated, etc.) then the header `"Content-`

`Length: number"` was not present in HTTP headers and the client assumed that when server closed the connection, the content had been entirely sent. This mechanism could not distinguish between a resource transfer successfully completed and an interrupted one (because of a server / network error or something else).

HTTP/1.1

HTTP/1.1 introduced:

- new headers to better manage the conditional retrieval of cached resources.

- chunked transfer encoding to allow content to be streamed in chunks in order to reliably send it even when the server does not know in advance its length (i.e. because it is dynamically generated, etc.).
- byte range serving, where a client can request only one or more portions (ranges of bytes) of a resource (i.e. the first part, a part in the middle or in the end of the entire content, etc.) and the server usually sends only the requested part(s). This is useful to resume an interrupted download (when a file is really big), when only a

part of a content has to be shown or dynamically added to the already visible part by a browser (i.e. only the first or the following n comments of a web page) in order to spare time, bandwidth and system resources, etc.

HTTP/2, HTTP/3

Both HTTP/2 and HTTP/3 have kept the above mentioned features of HTTP/1.1.

HTTP authentication

HTTP provides multiple authentication schemes such as basic access authentication and digest access authentication which operate via a challenge–response mechanism whereby the server identifies and issues a challenge before serving the requested content.

HTTP provides a general framework for access control and authentication, via an extensible set of challenge–response authentication schemes, which can be used by a server to challenge a client request and by a client to provide authentication information.[1]

The authentication mechanisms described above belong to the HTTP protocol and are managed by client and server HTTP software (if configured to require authentication before allowing client access to one or more web resources), and not by the web applications using a web application session.

Authentication realms

The HTTP Authentication specification also provides an arbitrary, implementation-specific construct for further dividing resources common to a given root URI. The realm value string, if present, is combined with the canonical root URI to form the protection space component of the challenge. This in effect allows the server to define separate authentication scopes under one root URI.[1]

HTTP application session

HTTP is a stateless protocol. A stateless protocol does not require the web server to retain information or status about each user for the duration of multiple requests.

Some web applications need to manage user sessions, so they implement states, or server side sessions, using for instance HTTP cookies[45] or hidden variables within web forms.

To start an application user session, an interactive authentication via web application login must be performed. To stop a user session a logout operation must be requested by user. These kind of operations do not use HTTP authentication but a custom managed web application authentication.

HTTP/1.1 request messages

Request messages are sent by a client to a target server.[note 4]

Request syntax

A client sends *request messages* to the server, which consist of:[46]

- a **request line**, consisting of the case-sensitive request method, a space, the requested URL, another space, the protocol version, a carriage return, and a line feed, e.g.:

```
GET /images/logo.png
HTTP/1.1
```

- zero or more request header fields (at least 1 or more headers in case of HTTP/1.1), each consisting of the case-insensitive field name, a colon, optional leading whitespace, the field value, an optional trailing whitespace and ending with a carriage return and a line feed, e.g.:

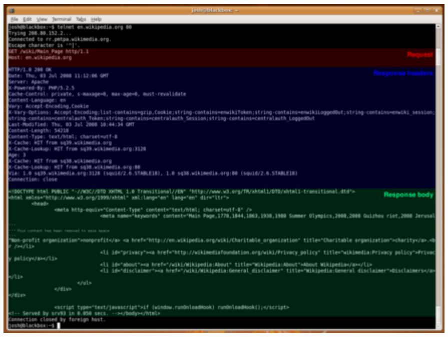
```
Host: www.example.com
Accept-Language: en
```

- an empty line, consisting of a carriage return and a line feed;
- an optional message body.

In the HTTP/1.1 protocol, all header fields except `Host: hostname` are optional.

A request line containing only the path name is accepted by servers to maintain compatibility with HTTP clients before the HTTP/1.0 specification in RFC 1945 (http s://datatracker.ietf.org/doc/html/rfc1945). .[47]

Request methods



```
josh@blackbox:~$ telnet en.wikipedia.org 80
Trying 208.80.152.2...
Connected to rr.pmtpa.wikimedia.org.
Escape character is '^]'.
GET /wiki/Main_Page http/1.1
Host: en.wikipedia.org

HTTP/1.0 200 OK
Date: Thu, 03 Jul 2008 11:12:06 GMT
Server: Apache
X-Powered-By: PHP/5.2.5
Cache-Control: private, s-maxage=0, max-age=0, must-revalidate
Content-Language: en
Vary: Accept-Encoding,Cookie
X-Vary-Options: Accept-Encoding;list-contains=gzip,Cookie;string-contains=enwikiToken;string-contains=enwikiLoggedOut;string-contains=enwiki_session;
string-contains=centralauth_Token;string-contains=centralauth_Session;string-contains=centralauth_LoggedOut
Last-Modified: Thu, 03 Jul 2008 10:44:34 GMT
Content-Length: 54218
Content-Type: text/html; charset=utf-8
X-Cache: HIT from sq39.wikimedia.org
X-Cache-Lookup: HIT from sq39.wikimedia.org:3128
Age: 3
X-Cache: HIT from sq38.wikimedia.org
X-Cache-Lookup: HIT from sq38.wikimedia.org:80
Via: 1.0 sq39.wikimedia.org:3128 (squid/2.6.STABLE18), 1.0 sq38.wikimedia.org:80 (squid/2.6.STABLE18)
Connection: close

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en" dir="ltr">
        <head>
                <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
                                <meta name="keywords" content="Main Page,1778,1844,1863,1938,1980 Summer Olympics,2008,2008 Guizhou riot,2008 Jerusal
...
... This content has been removed to save space
...
"Non-profit organization">nonprofit</a> <a href="http://en.wikipedia.org/wiki/Charitable_organization" title="Charitable organization">charity</a>.<b
r /></li>
                                <li id="privacy"><a href="http://wikimediafoundation.org/wiki/Privacy_policy" title="wikimedia:Privacy policy">Privac
y policy</a></li>
                                <li id="about"><a href="/wiki/Wikipedia:About" title="Wikipedia:About">About Wikipedia</a></li>
                                <li id="disclaimer"><a href="/wiki/Wikipedia:General_disclaimer" title="Wikipedia:General disclaimer">Disclaimers</a>
</li>
                        </ul>
                </div>
</div>

                <script type="text/javascript">if (window.runOnloadHook) runOnloadHook();</script>
<!-- Served by srv93 in 0.050 secs. --></body></html>
Connection closed by foreign host.
josh@blackbox:~$
```

An HTTP/1.1 request made using telnet. The request message, response header section, and response body are highlighted.

HTTP defines methods (sometimes referred to as *verbs*, but nowhere in the specification does it mention *verb*) to indicate the desired action to be performed on the identified resource. What this resource represents, whether preexisting data or data that is generated dynamically, depends on the implementation of the server. Often, the resource corresponds to a file or the output of an executable residing on the server. The HTTP/1.0 specification[48] defined the GET, HEAD, and POST methods, and the HTTP/1.1 specification[49] added five new methods: PUT, DELETE, CONNECT, OPTIONS, and TRACE. Any client can use any method and the server can be configured to support any combination of methods. If a method is unknown to an intermediate, it will be treated as an unsafe and non-idempotent method. There is no limit to the number of methods that can be defined, which allows for future methods to be specified without breaking existing infrastructure. For example, WebDAV defined seven new methods and RFC 5789 (https://datatracker.ietf.org/doc/html/rfc5789) specified the PATCH method.

Method names are case sensitive.[50][51] This is in contrast to HTTP header field names which are case-insensitive.[52]

GET

The GET method requests that the target resource transfer a representation of its state. GET requests should only retrieve data and should have no other effect. (This is also true of some other HTTP methods.)[1] For retrieving resources without making changes, GET is preferred over POST, as they can be addressed through a URL. This enables bookmarking and sharing and makes GET responses eligible for caching, which can save bandwidth. The W3C has published guidance principles on this distinction, saying, "Web application design should be informed by the above principles, but also by the relevant limitations."[53] See safe methods below.

HEAD

The HEAD method requests that the target resource transfer a representation of its state, as for a GET request, but without the representation data enclosed in the response body. This is useful for retrieving the representation metadata in the response header, without having to transfer the entire representation. Uses include checking whether a page is available through the status code and quickly finding the size of a file (`Content-Length`).

POST

The POST method requests that the target resource process the representation enclosed in the request according to the semantics of the target resource. For example, it is used for posting a message to an Internet forum, subscribing to a mailing list, or completing an online shopping transaction.[54]

PUT

The PUT method requests that the target resource create or update its state with the state defined by the representation enclosed in the request. A distinction from POST is that the client specifies the target location on the server.[55]

DELETE

The DELETE method requests that the target resource delete its state.

CONNECT

The CONNECT method requests that the intermediary establish a TCP/IP tunnel to the origin server identified by the request target. It is often used to secure connections through one or more HTTP proxies with TLS.[56][57] See HTTP CONNECT method.

OPTIONS

The OPTIONS method requests that the target resource transfer the HTTP methods that it supports. This can be used to check the functionality of a web server by requesting '*' instead of a specific resource.

TRACE

The TRACE method requests that the target resource transfer the received request in the response body. That way a client can see what (if any) changes or additions have been made by intermediaries.

PATCH

The PATCH method requests that the target resource modify its state according to the partial update defined in the representation enclosed in the request. This can save bandwidth by updating a part of a file or document without having to transfer it entirely.[58]

All general-purpose web servers are required to implement at least the GET and HEAD methods, and all other methods are considered optional by the specification.[51]

Properties of request methods

Request method	RFC	Request has payload body	Response has payload body	Safe	Idempotent	Cacheable
GET	RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110)	Optional	Yes	Yes	Yes	Yes
HEAD	RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110)	Optional	No	Yes	Yes	Yes
POST	RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110)	Yes	Yes	No	No	Yes
PUT	RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110)	Yes	Yes	No	Yes	No
DELETE	RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110)	Optional	Yes	No	Yes	No
CONNECT	RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110)	Optional	Yes	No	No	No
OPTIONS	RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110)	Optional	Yes	Yes	Yes	No

Request method	RFC	Request has payload body	Response has payload body	Safe	Idempotent	Cacheable
TRACE	RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110)	No	Yes	Yes	Yes	No
PATCH	RFC 5789 (https://datatracker.ietf.org/doc/html/rfc5789)	Yes	Yes	No	No	No

Safe methods

A request method is *safe* if a request with that method has no intended effect on the server. The methods GET, HEAD, OPTIONS, and TRACE are defined as safe. In other words, safe methods are intended to be read-only. They do not exclude side effects though, such as appending request information to a log file or charging an advertising account, since they are not requested by the client, by definition.

In contrast, the methods POST, PUT, DELETE, CONNECT, and PATCH are not safe. They may modify the state of the server or have other effects such as sending an email. Such methods are therefore not usually used by conforming web robots or web crawlers; some that do not conform tend to make requests without regard to context or consequences.

Despite the prescribed safety of GET requests, in practice their handling by the server is not technically limited in any way. Careless or deliberately irregular programming can allow GET requests to cause non-trivial changes on the server. This is discouraged because of the problems which can occur when web caching, search engines, and other automated agents make unintended changes on the server. For example, a website might allow deletion of a resource through a URL such as *https://example.com/article/1234/delete*, which, if arbitrarily fetched, even using GET, would simply delete the article.[59] A properly coded website would require a

DELETE or POST method for this action, which non-malicious bots would not make.

One example of this occurring in practice was during the short-lived Google Web Accelerator beta, which prefetched arbitrary URLs on the page a user was viewing, causing records to be automatically altered or deleted *en masse*. The beta was suspended only weeks after its first release, following widespread criticism.[60][59]

Idempotent methods

A request method is *idempotent* if multiple identical requests with that method have the same effect as a single such request. The methods PUT and DELETE, and safe methods are defined as idempotent. Safe methods are trivially idempotent, since they are intended to have no effect on the server whatsoever; the PUT and DELETE methods, meanwhile, are idempotent since successive identical requests will be ignored. A website might, for instance, set up a PUT endpoint to modify a user's recorded email address. If this endpoint is configured correctly, any requests which ask to change a user's email address to the same email address which is already recorded—e.g. duplicate requests following a successful request—will have no effect. Similarly, a request to DELETE a certain user will have no effect if that user has already been deleted.

In contrast, the methods POST, CONNECT, and PATCH are not necessarily idempotent, and therefore sending an identical POST request multiple times may further modify the state of the server or have further effects, such as sending multiple emails. In some cases this is the desired effect, but in other cases it may occur accidentally. A user might, for example, inadvertently send multiple POST requests by clicking a button again if they were not given clear feedback that the first click was being processed. While web browsers may show alert dialog boxes to warn users in some cases where reloading a page may re-submit a POST request, it is generally up to the web application to handle cases where a POST request should not be submitted more than once.

Note that whether or not a method is idempotent is not enforced by the protocol or web server. It is perfectly possible to write a web application in which (for example) a database insert or other non-idempotent action is triggered by a GET or other request. To do so against recommendations, however, may result in undesirable consequences, if a user agent assumes that repeating the same request is safe when it is not.

Cacheable methods

A request method is *cacheable* if responses to requests with that method may be stored for future reuse. The methods GET, HEAD, and POST are defined as cacheable.

In contrast, the methods PUT, DELETE, CONNECT, OPTIONS, TRACE, and PATCH are not cacheable.

Request header fields

Request header fields allow the client to pass additional information beyond the request line, acting as request modifiers (similarly to the parameters of a procedure). They give information about the client, about the target resource, or about the expected handling of the request.

HTTP/1.1 response messages

A response message is sent by a server to a client as a reply to its former request message.[note 4]

Response syntax

A server sends *response messages* to the client, which consist of:[46]

- a **status line**, consisting of the protocol version, a space, the response status code, another space, a possibly empty reason phrase, a carriage return and a line feed, e.g.:

```
HTTP/1.1 200 OK
```

- zero or more response header fields, each consisting of the case-insensitive field name, a colon, optional leading whitespace, the field value, an optional trailing whitespace and ending with a carriage return and a line feed, e.g.:

```
Content-Type: text/html
```

- an empty line, consisting of a carriage return and a line feed;
- an optional message body.

Response status codes

In HTTP/1.0 and since, the first line of the HTTP response is called the *status line* and includes a numeric *status code* (such as "404") and a textual *reason phrase* (such as "Not Found"). The response status code is a three-digit integer code representing the result of the server's attempt to understand and satisfy the client's corresponding request. The way the client handles the response depends primarily on the status code, and secondarily on the other response header fields. Clients may not understand all registered status codes but they must understand their class (given by the first digit of the status code) and treat an unrecognized status code as being equivalent to the x00 status code of that class.

The standard *reason phrases* are only recommendations, and can be replaced with "local equivalents" at the web developer's discretion. If the status code indicated a problem, the user agent might display the *reason phrase* to the user to provide further information about the nature of the problem. The standard also allows the user agent to attempt to interpret the *reason phrase*, though this might be unwise since the standard explicitly specifies that status codes are machine-readable and *reason phrases* are human-readable.

The first digit of the status code defines its class:

`1XX` **(informational)**

The request was received, continuing process.

`2XX` **(successful)**

The request was successfully received, understood, and accepted.

`3XX` **(redirection)**

Further action needs to be taken in order to complete the request.

`4XX` **(client error)**

The request contains bad syntax or cannot be fulfilled.

`5XX` **(server error)**

The server failed to fulfill an apparently valid request.

Response header fields

The response header fields allow the server to pass additional information beyond the status line, acting as response modifiers. They give information about the server or about further access to the target resource or related resources.

Each response header field has a defined meaning which can be further refined by the semantics of the request method or response status code.

HTTP/1.1 example of request / response transaction

Below is a sample HTTP transaction between an HTTP/1.1 client and an HTTP/1.1 server running on www.example.com, port 80.[note 5][note 6]

Client request

```
GET / HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept:
text/html,application/xhtml
```

```
+xml,application/xml;q=0.9,
image/avif,image/webp,*/*;q
=0.8
Accept-Language: en-
GB,en;q=0.5
Accept-Encoding: gzip,
deflate, br
Connection: keep-alive
```

A client request (consisting in this case of the request line and a few headers that can be reduced to only the `"Host: hostname"` header) is followed by a blank line, so that the request ends with a double end of line, each in the form of a carriage return followed by a line feed. The

`"Host: hostname"` header value distinguishes between various DNS names sharing a single IP address, allowing name-based virtual hosting. While optional in HTTP/1.0, it is mandatory in HTTP/1.1. (A "/" (slash) will usually fetch a /index.html file if there is one.)

Server response

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005
22:38:34 GMT
Content-Type: text/html;
charset=UTF-8
```

```
Content-Length: 155
Last-Modified: Wed, 08 Jan
2003 23:11:55 GMT
Server: Apache/1.3.3.7
(Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
  <head>
    <title>An Example
Page</title>
  </head>
  <body>
    <p>Hello World, this is
```

```
a very simple HTML
document.</p>
  </body>
</html>
```

The ETag (entity tag) header field is used to determine if a cached version of the requested resource is identical to the current version of the resource on the server. `"Content-Type"` specifies the Internet media type of the data conveyed by the HTTP message, while `"Content-Length"` indicates its length in bytes. The HTTP/1.1 webserver publishes its ability to respond to requests for certain byte ranges of the document by setting the field `"Accept-Ranges: bytes"`. This is useful, if the client needs to have only certain portions[61] of a resource sent by the server, which is called byte serving. When `"Connection: close"` is sent, it means that the web server will close the TCP connection immediately after the end of the transfer of this response.[21]

Most of the header lines are optional but some are mandatory. When header `"Content-Length: number"` is missing in a response with an entity body then this should be considered an error in HTTP/1.0 but it may not be an error in HTTP/1.1 if header `"Transfer-`

`Encoding: chunked"` is present. Chunked transfer encoding uses a chunk size of 0 to mark the end of the content. Some old implementations of HTTP/1.0 omitted the header `"Content-Length"` when the length of the body entity was not known at the beginning of the response and so the transfer of data to client continued until server closed the socket.

A `"Content-Encoding: gzip"` can be used to inform the client that the body entity part of the transmitted data is compressed by gzip algorithm.

Encrypted connections

The most popular way of establishing an encrypted HTTP connection is HTTPS.[62] Two other methods for establishing an encrypted HTTP connection also exist: Secure Hypertext Transfer Protocol, and using the HTTP/1.1 Upgrade header to specify an upgrade to TLS. Browser support for these two is, however, nearly non-existent.[63][64][65]

Similar protocols

- The Gopher protocol is a content delivery protocol that was displaced by

HTTP in the early 1990s.

- The SPDY protocol is an alternative to HTTP developed at Google, superseded by HTTP/2.
- The Gemini protocol is a Gopher-inspired protocol which mandates privacy-related features.

See also

- InterPlanetary File System - can replace http
- Comparison of file transfer protocols
- Constrained Application Protocol – a semantically similar protocol to HTTP but used UDP or UDP-like messages

targeted for devices with limited processing capability; re-uses HTTP and other internet concepts like Internet media type and web linking (RFC 5988)[66]

- Content negotiation
- Digest access authentication
- HTTP compression
- HTTP/2 – developed by the IETF's Hypertext Transfer Protocol (httpbis) working group[35]
- List of HTTP header fields
- List of HTTP status codes
- Representational state transfer (REST)

- Variant object
- Web cache
- WebSocket

Notes

1. *In practice, these streams are used as multiple TCP/IP sub-connections to* multiplex *concurrent requests/responses, thus greatly reducing the number of real TCP/IP connections on server side, from 2..8 per client to 1, and allowing many more clients to be served at once.*
2. *In 2022, HTTP/0.9 support has not been officially completely deprecated and is still present in many web servers and browsers (for server responses only), even if usually*

disabled. It is unclear how long it will take to decommission HTTP/0.9.

3. *Since late 1996, some developers of popular HTTP/1.0 browsers and servers (specially those who had planned support for HTTP/1.1 too), started to deploy (as an unofficial extension) a sort of keep-alive-mechanism (by using new HTTP headers) in order to keep the TCP/IP connection open for more than a request/response pair and so to speed up the exchange of multiple requests/responses.*[31]
4. *HTTP/2 and HTTP/3 have a different representation for HTTP methods and headers.*
5. *HTTP/1.0 has the same messages except for a few missing headers.*

6. *HTTP/2 and HTTP/3 use the same request / response mechanism but with different representations for HTTP headers.*

References

1. *Fielding, R.; Nottingham, M.; Reschke, J. (June 2022). HTTP Semantics (https://datatracker.ietf.org/doc/html/rfc9110) . IETF. doi:10.17487/RFC9110 (https://doi.org/10.17487%2FRFC9110) . RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110) .*
2. *Tim Berner-Lee (1991-01-01). "The Original HTTP as defined in 1991" (https://www.w3.org/pub/WWW/Protocols/HTTP/AsImplemented.html) . www.w3.org. World Wide Web Consortium. Retrieved 2010-07-24.*

3. *Tim Berner-Lee (1992). "Basic HTTP as defined in 1992" (https://www.w3.org/Protocols/HTTP/HTTP2.html) . www.w3.org. World Wide Web Consortium. Retrieved 2021-10-19.*
4. *In RFC 1945 (https://datatracker.ietf.org/doc/html/rfc1945) . That specification was then overcome by HTTP/1.1.*
5. *RFC 2068 (https://datatracker.ietf.org/doc/html/rfc2068) (1997) was obsoleted by RFC 2616 (https://datatracker.ietf.org/doc/html/rfc2616) in 1999, which was obsoleted by RFC 7230 (https://datatracker.ietf.org/doc/html/rfc7230) in 2014, which was obsoleted by RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110) in 2022.*

6. *"Usage Statistics of Default protocol https for websites" (https://w3techs.com/technologies/details/ce-httpsdefault) . w3techs.com. Retrieved 2022-10-16.*

7. *"Usage Statistics of HTTP/2 for websites" (https://w3techs.com/technologies/details/ce-http2) . w3techs.com. Retrieved 2023-04-20.*

8. *"Can I use... Support tables for HTML5, CSS3, etc" (https://caniuse.com/?search=http2) . caniuse.com. Retrieved 2022-10-16.*

9. *Friedl, S.; Popov, A.; Langley, A.; Stephan, E. (July 2014). Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension (https://datatracker.ietf.org/doc/html/rfc7301) . IETF. doi:10.17487/RFC7301 (https://doi.org/10.17487%2FRFC7301) . RFC 7301 (https://datatracker.ietf.org/doc/html/rfc7301) .*

10. *Belshe, M.; Peon, R.; Thomson, M. "Hypertext Transfer Protocol Version 2, Use of TLS Features" (https://web.archive.org/web/20130715004452/https://http2.github.io/http2-spec/#TLSUsage) . Archived from the original (https://http2.github.io/http2-spec/#TLSUsage) on 2013-07-15. Retrieved 2015-02-10.*

11. *Benjamin, David. Using TLS 1.3 with HTTP/2 (https://datatracker.ietf.org/doc/html/rfc8740) . doi:10.17487/RFC8740 (https://doi.org/10.17487%2FRFC8740) . RFC 8740 (https://datatracker.ietf.org/doc/html/rfc8740) . Retrieved 2020-06-02. "This lowers the barrier for deploying TLS 1.3, a major security improvement over TLS 1.2."*

12. *HTTP/3 (https://datatracker.ietf.org/doc/html/rfc9114) . 6 June 2022. doi:10.17487/RFC9114 (https://doi.org/10.17487%2FRFC9114) . RFC 9114 (https://datatracker.ietf.org/doc/html/rfc9114) . Retrieved 2022-06-06.*

13. *"Usage Statistics of HTTP/3 for websites" (https://w3techs.com/technologies/details/ce-http3) . w3techs.com. Retrieved 2023-07-20.*

14. *"Can I use... Support tables for HTML5, CSS3, etc" (https://caniuse.com/?search=http3) . caniuse.com. Retrieved 2023-07-20.*

15. *Cimpanu, Catalin (26 September 2019). "Cloudflare, Google Chrome, and Firefox add HTTP/3 support" (https://www.zdnet.com/article/cloudflare-google-chrome-and-firefox-add-http3-support/) . ZDNet. Retrieved 27 September 2019.*

16. *"HTTP/3: the past, the present, and the future" (https://blog.cloudflare.com/http3-the-past-present-and-future/) . The Cloudflare Blog. 2019-09-26. Retrieved 2019-10-30.*

17. *"Firefox Nightly supports HTTP 3 – General – Cloudflare Community" (https://community.cloudflare.com/t/firefox-nightly-supports-http-3/127778) . 2019-11-19. Retrieved 2020-01-23.*

18. *"HTTP/3 is Fast" (https://requestmetrics.com/web-performance/http3-is-fast) . Request Metrics. Retrieved 2022-07-01.*

19. *"Connections, Clients, and Servers" (https://datatracker.ietf.org/doc/html/rfc9110#section-3.3) . RFC 9110, HTTP Semantics (https://datatracker.ietf.org/doc/html/rfc9110) . sec. 3.3. doi:10.17487/RFC9110 (https://doi.org/10.17487%2FRFC9110) . RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110) .*

20. *"Overall Operation" (https://datatracker.ietf.org/doc/html/rfc1945#section-1.3) . RFC 1945 (https://datatracker.ietf.org/doc/html/rfc1945) . pp. 6–8. sec. 1.3. doi:10.17487/RFC1945 (https://doi.org/10.17487%2FRFC1945) . RFC 1945 (https://datatracker.ietf.org/doc/html/rfc1945) .*

21. "Connection Management: Establishment" (https://datatracker.ietf.org/doc/html/rfc9112#section-9.1) . RFC 9112, HTTP/1.1 (https://datatracker.ietf.org/doc/html/rfc9112) . sec. 9.1. doi:10.17487/RFC9112 (https://doi.org/10.17487%2FRFC9112) . RFC 9112 (https://datatracker.ietf.org/doc/html/rfc9112) .
22. "Connection Management: Persistence" (https://datatracker.ietf.org/doc/html/rfc9112#section-9.3) . RFC 9112, HTTP/1.1 (https://datatracker.ietf.org/doc/html/rfc9112) . sec. 9.3. doi:10.17487/RFC9112 (https://doi.org/10.17487%2FRFC9112) . RFC 9112 (https://datatracker.ietf.org/doc/html/rfc9112) .

23. *"Classic HTTP Documents" (https://www.w3.org/Protocols/Classic.html) . W3.org. 1998-05-14. Retrieved 2010-08-01.*

24. *"HTTP/2 Protocol Overview" (https://datatracker.ietf.org/doc/html/rfc7540#section-2) . RFC 9113, HTTP/2) (https://datatracker.ietf.org/doc/html/rfc7540) . sec. 2. doi:10.17487/RFC7540 (https://doi.org/10.17487%2FRFC7540) . RFC 7540 (https://datatracker.ietf.org/doc/html/rfc7540) .*

25. *"Invention Of The Web, Web History, Who Invented the Web, Tim Berners-Lee, Robert Cailliau, CERN, First Web Server" (https://www.livinginternet.com/w/wi_lee.htm) . LivingInternet. Retrieved 2021-08-11.*

26. *Berners-Lee, Tim (1990-10-02). "daemon.c - TCP/IP based server for HyperText" (https://www.w3.org/Daemon/old/V0.1/daemon.c) . www.w3.org. Retrieved 2021-08-11.*

27. *Berners-Lee, Tim. "HyperText Transfer Protocol" (https://www.w3.org/History/19921103-hypertext/hypertext/WWW/Protocols/HTTP.html) . World Wide Web Consortium. Retrieved 31 August 2010.*

28. *Raggett, Dave. "Dave Raggett's Bio" (https://www.w3.org/People/Raggett/profile.html) . World Wide Web Consortium. Retrieved 11 June 2010.*

29. *Raggett, Dave; Berners-Lee, Tim. "Hypertext Transfer Protocol Working Group" (https://www.w3.org/Arena/webworld/httpwgcharter.html) . World Wide Web Consortium. Retrieved 29 September 2010.*

30. *Raggett, Dave. "HTTP WG Plans" (https://www.w3.org/Arena/webworld/httpwgplans.html) . World Wide Web Consortium. Retrieved 29 September 2010.*

31. *David Gourley; Brian Totty; Marjorie Sayer; Anshu Aggarwal; Sailu Reddy (2002). HTTP: The Definitive Guide. (excerpt of chapter: "Persistent Connections") (https://www.oreilly.com/library/view/http-the-definitive/1565925092/ch04s05.html) . O'Reilly Media, inc. ISBN 9781565925090. Retrieved 2021-10-18.*

32. "HTTP/1.1" (http://webarchive.loc.gov/all/20011121001051/https://www.webcom.com/glossary/http1.1.shtml) . Webcom.com Glossary entry. Archived from the original (https://www.webcom.com/glossary/http1.1.shtml) on 2001-11-21. Retrieved 2009-05-29.
33. "HTTP-NG Working Group" (https://www.w3.org/Protocols/HTTP-NG/) . www.w3.org. World Wide Web Consortium. 1997. Retrieved 2021-10-19.
34. Web Administrator (2007). "HTTP Working Group" (https://httpwg.org/) . httpwg.org. IETF. Retrieved 2021-10-19.

35. *Web Administrator (2007). "HTTP Working Group: charter httpbis" (https://datatracker.ietf.org/wg/httpbis/charter/) . datatracker.ietf.org. IETF. Retrieved 2021-10-19.*

36. *"SPDY: An experimental protocol for a faster web" (http://dev.chromium.org/spdy/spdy-whitepaper) . dev.chromium.org. Google. 2009-11-01. Retrieved 2021-10-19.*

37. *"Rechartering httpbis" (https://lists.w3.org/Archives/Public/ietf-http-wg/2012JanMar/0098.html) . IETF; HTTP WG. 2012-01-24. Retrieved 2021-10-19.*

38. *IESG Secretary (2012-03-19). "WG Action: RECHARTER: Hypertext Transfer Protocol Bis (httpbis)" (https://lists.w3.org/Archives/Public/ietf-http-wg/2012JanMar/0902.html) . IETF; HTTP WG. Retrieved 2021-10-19.*

39. *Ilya Grigorik; Surma (2019-09-03). "High Performance Browser Networking: Introduction to HTTP/2" (https://developers.google.com/web/fundamentals/performance/http2) . developers.google.com. Google Inc. Retrieved 2021-10-19.*

40. *"Appendix-A: HTTP Version History" (https://datatracker.ietf.org/doc/html/rfc7230#appendix-A) . RFC 7230, HTTP/1.1: Message Syntax and Routing (https://datatracker.ietf.org/doc/html/rfc7230) . p. 78. sec. A. doi:10.17487/RFC7230 (https://doi.org/10.17487%2FRFC7230) . RFC 7230 (https://datatracker.ietf.org/doc/html/rfc7230) .*

41. *Matt Menke (2016-06-30). "Intent to Deprecate and Remove: HTTP/0.9 Support" (https://groups.google.com/a/chromium.org/g/blink-dev/c/OdKnpLlvVUo/m/1EpFGVUjAwAJ) . groups.google.com. Retrieved 2021-10-15.*

42. *HTTP/3 (https://datatracker.ietf.org/doc/html/rfc9114) . 6 June 2022. doi:10.17487/RFC9114 (https://doi.org/10.17487%2FRFC9114) . RFC 9114 (https://datatracker.ietf.org/doc/html/rfc9114) . Retrieved 2022-06-06.*

43. *"http URI Scheme" (https://datatracker.ietf.org/doc/html/rfc9110#section-4.2.1) . RFC 9110, HTTP Semantics (https://datatracker.ietf.org/doc/html/rfc9110) . sec. 4.2.1. doi:10.17487/RFC9110 (https://doi.org/10.17487%2FRFC9110) . RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110) .*

44. *"https URI Scheme" (https://datatracker.ietf.org/doc/html/rfc9110#section-4.2.2) . RFC 9110, HTTP Semantics (https://datatracker.ietf.org/doc/html/rfc9110) . sec. 4.2.2. doi:10.17487/RFC9110 (https://doi.org/10.17487%2FRFC9110) . RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110) .*

45. Lee, Wei-Bin; Chen, Hsing-Bai; Chang, Shun-Shyan; Chen, Tzung-Her (2019-01-25). "Secure and efficient protection for HTTP cookies with self-verification" (https://onlinelibrary.wiley.com/doi/10.1002/dac.3857) . International Journal of Communication Systems. **32** (2): e3857. doi:10.1002/dac.3857 (https://doi.org/10.1002%2Fdac.3857) . S2CID 59524143 (https://api.semanticscholar.org/CorpusID:59524143) .

46. "Message format" (https://datatracker.ietf.org/doc/html/rfc9112#section-2.1) . RFC 9112: HTTP/1.1 (https://datatracker.ietf.org/doc/html/rfc9112) . sec. 2.1. doi:10.17487/RFC9112 (https://doi.org/10.17487%2FRFC9112) . RFC 9112 (https://datatracker.ietf.org/doc/html/rfc9112) .
47. "Apache Week. HTTP/1.1" (https://www.apacheweek.com/features/http11) . 090502 apacheweek.com

48. *Berners-Lee, Tim; Fielding, Roy T.; Nielsen, Henrik Frystyk. "Method Definitions" (https://datatracker.ietf.org/doc/html/rfc1945#section-8) . Hypertext Transfer Protocol – HTTP/1.0 (https://datatracker.ietf.org/doc/html/rfc1945) . IETF. pp. 30–32. sec. 8. doi:10.17487/RFC1945 (https://doi.org/10.17487%2FRFC1945) . RFC 1945 (https://datatracker.ietf.org/doc/html/rfc1945) .*

49. *"Method Definitions" (https://datatracker.ietf.org/doc/html/rfc2616#section-9) . RFC 2616 (https://datatracker.ietf.org/doc/html/rfc2616) . pp. 51–57. sec. 9. doi:10.17487/RFC2616 (https://doi.org/10.17487%2FRFC2616) . RFC 2616 (https://datatracker.ietf.org/doc/html/rfc2616) .*

50. *"Request Line" (https://datatracker.ietf.org/doc/html/rfc9112#section-3) . RFC 9112, HTTP/1.1 (https://datatracker.ietf.org/doc/html/rfc9112) . sec. 3. doi:10.17487/RFC9112 (https://doi.org/10.17487%2FRFC9112) . RFC 9112 (https://datatracker.ietf.org/doc/html/rfc9112) .*

51. *"Methods: Overview" (https://datatracker.ietf.org/doc/html/rfc9110#section-9.1) . RFC 9110, HTTP Semantics (https://datatracker.ietf.org/doc/html/rfc9110) . sec. 9.1. doi:10.17487/RFC9110 (https://doi.org/10.17487%2FRFC9110) . RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110) .*

52. *"Header Fields" (https://datatracker.ietf.org/doc/html/rfc9110#section-6.3) . RFC 9110, HTTP Semantics (https://datatracker.ietf.org/doc/html/rfc9110) . sec. 6.3. doi:10.17487/RFC9110 (https://doi.org/10.17487%2FRFC9110) . RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110) .*

53. *Jacobs, Ian (2004). "URIs, Addressability, and the use of HTTP GET and POST" (https://www.w3.org/2001/tag/doc/whenToUseGet.html#checklist) . Technical Architecture Group finding. W3C. Retrieved 26 September 2010.*

54. *"POST" (https://datatracker.ietf.org/doc/html/rfc9110#section-9.3.3) . RFC 9110, HTTP Semantics (https://datatracker.ietf.org/doc/html/rfc9110) . sec. 9.3.3. doi:10.17487/RFC9110 (https://doi.org/10.17487%2FRFC9110) . RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110) .*

55. *"PUT" (https://datatracker.ietf.org/doc/html/rfc9110#section-9.3.4) . RFC 9110, HTTP Semantics (https://datatracker.ietf.org/doc/html/rfc9110) . sec. 9.3.4. doi:10.17487/RFC9110 (https://doi.org/10.17487%2FRFC9110) . RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110) .*

56. "CONNECT" (https://datatracker.ietf.org/doc/html/rfc9110#section-9.3.6) . RFC 9110, HTTP Semantics (https://datatracker.ietf.org/doc/html/rfc9110) . sec. 9.3.6. doi:10.17487/RFC9110 (https://doi.org/10.17487%2FRFC9110) . RFC 9110 (https://datatracker.ietf.org/doc/html/rfc9110) .

57. "Vulnerability Note VU#150227: HTTP proxy default configurations allow arbitrary TCP connections" (https://www.kb.cert.org/vuls/id/150227) . US-CERT. 2002-05-17. Retrieved 2007-05-10.

58. Dusseault, Lisa; Snell, James M. (March 2010). PATCH Method for HTTP (https://datatracker.ietf.org/doc/html/rfc5789) . IETF. doi:10.17487/RFC5789 (https://doi.org/10.17487%2FRFC5789) . RFC 5789 (https://datatracker.ietf.org/doc/html/rfc5789) .

59. Ediger, Brad (2007-12-21). Advanced Rails: Building Industrial-Strength Web Apps in Record Time (https://shop.oreilly.com/product/9780596510329.do) . O'Reilly Media, Inc. p. 188. ISBN 978-0596519728. "A common mistake is to use GET for an action that updates a resource. [...] This problem came into the Rails public eye in 2005, when the Google Web Accelerator was released."

60. *Cantrell, Christian (2005-06-01). "What Have We Learned From the Google Web Accelerator?" (https://web.archive.org/web/20170819161233/https://blogs.adobe.com/cantrell/archives/2005/06/what_have_we_le.html) . Adobe Blogs. Adobe. Archived from the original (https://blogs.adobe.com/cantrell/archives/2005/06/what_have_we_le.html) on 2017-08-19. Retrieved 2018-11-19.*

61. *Luotonen, Ari; Franks, John (February 22, 1996). Byte Range Retrieval Extension to HTTP (https://datatracker.ietf.org/doc/html/draft-ietf-http-range-retrieval-00) . IETF. I-D draft-ietf-http-range-retrieval-00.*

62. Canavan, John (2001). Fundamentals of Networking Security. Norwood, MA: Artech House. pp. 82–83. ISBN 9781580531764.

63. Zalewski, Michal. "Browser Security Handbook" (https://code.google.com/p/browsersec/wiki/Part1#True_URL_schemes) . Retrieved 30 April 2015.

64. "Chromium Issue 4527: implement RFC 2817: Upgrading to TLS Within HTTP/1.1" (https://code.google.com/p/chromium/issues/detail?id=4527) . Retrieved 30 April 2015.

65. "Mozilla Bug 276813 – [RFE] Support RFC 2817 / TLS Upgrade for HTTP 1.1" (https://bugzilla.mozilla.org/show_bug.cgi?id=276813) . Retrieved 30 April 2015.

66. *Nottingham, Mark (October 2010). Web Linking (https://datatracker.ietf.org/doc/html/rfc5988) . IETF. doi:10.17487/RFC5988 (https://doi.org/10.17487%2FRFC5988) . RFC 5988 (https://datatracker.ietf.org/doc/html/rfc5988) .*

External links

- Official website (https://httpwg.org/specs/)
- IETF HTTP Working Group (https://github.com/httpwg) on GitHub
- "Change History for HTTP" (https://www.w3.org/Protocols/History.html) . W3.org.

Wikimedia Commons has media related to ***Hypertext***

Retrieved 2010-08-01. A detailed technical history of HTTP.

Transfer Protocol.

- "Design Issues for HTTP" (https://www.w3.org/Protocols/DesignIssues.html) . W3.org. Retrieved 2010-08-01. Design Issues by Berners-Lee when he was designing the protocol.

Retrieved from "https://en.wikipedia.org/w/index.php?title=HTTP&oldid=1178865608"