

The Wayback Machine - <https://web.archive.org/web/20060323170107/http://en.wikipedia.org/wiki/HTTP>

HyperText Transfer Protocol

From Wikipedia, the free encyclopedia
(Redirected from HTTP)

HyperText Transfer Protocol (HTTP) is the method used to transfer or convey information on the World Wide Web. The original purpose was to provide a way to publish and receive HTML pages.

Development of HTTP was coordinated by the World Wide Web Consortium and working groups of the Internet Engineering Task Force, culminating in the publication of a series of RFCs, most notably RFC 2616, which defines HTTP/1.1, the version of HTTP in common use today.

HTTP is a request/response protocol between clients and servers. The originating client, such as a web browser, spider, or other end-user tool, is referred to as the user agent. The destination server, which stores or creates resources such as HTML files and images, is called the origin server. Inbetween the user agent and origin server may be several intermediaries, such as proxies, gateways, and tunnels.

A HTTP client initiates a request by establishing a Transmission Control Protocol (TCP) connection to a particular port on a remote host (port 80 by default; see List of well-known ports (computing)). A HTTP server listening on that port waits for the client to send a Request Message.

Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message of its own, the body of which is perhaps the requested file, an error message, or some other information.

Resources to be accessed by HTTP are identified using Uniform Resource Identifiers (URIs) (or, more specifically, URLs) using the `http:` or `https:` URI schemes.

Internet protocol suite

Layer	Protocols
Application	DNS, TLS/SSL, TFTP, FTP, HTTP , IMAP, IRC, NNTP, POP3, SIP, SMTP, SNMP, SSH, TELNET, BitTorrent, RTP, rlogin, ENRP, ...
Transport	TCP, UDP, DCCP, SCTP, IL, RUDP, ...
Network	IP (IPv4, IPv6), ICMP, IGMP, ARP, RARP, ...
Link	Ethernet, Wi-Fi, Token ring, PPP, SLIP, FDDI, ATM, Frame Relay, SMDS, ...

Contents

- 1 Request Message
- 2 Request methods
- 3 HTTP versions
- 4 Status codes
- 5 HTTP connection persistence
- 6 HTTP connection closing
- 7 HTTP session state
- 8 Secure HTTP
- 9 Sample
- 10 See also

- 11 External links
 - 11.1 Specifications and references
 - 11.2 Tutorials and tools

Request Message

The request message consists of the following:

- Request line, such as `GET /images/logo.gif HTTP/1.1`, which requests the file `logo.gif` from the `/images` directory
- Headers, such as `Accept-Language: en`
- An empty line
- An optional message body

The request line and headers must all end with CRLF (*i.e.* a carriage return followed by a line feed). The empty line must consist of only CRLF and no other whitespace.

Some headers are optional, while others (such as `Host`) are required by the HTTP/1.1 protocol.

Request methods

HTTP defines eight methods indicating the desired action to be performed on the identified resource.

- **GET** – Requests a representation of the specified resource. By far the most common method used on the Web today.
- **HEAD** – Asks for the response identical to the one that would correspond to a GET request, but without the response body. This is useful for retrieving meta-information written in response headers, without having to transport the entire content.
- **POST** – Submits user data (e.g. from a HTML form) to the identified resource. The data is included in the body of the request.
- **PUT** – Uploads a representation of the specified resource.
- **DELETE** – Deletes the specified resource (rarely implemented).
- **TRACE** – Echoes back the received request, so that a client can see what intermediate servers are adding or changing in the request.
- **OPTIONS** – Returns the HTTP methods that the server supports. This can be used to check the functionality of a web server.
- **CONNECT** – For use with a proxy that can change to being an SSL tunnel.

Methods GET and HEAD are defined as *safe*, i.e. intended only for information retrieval. Unsafe methods (such as POST, PUT and DELETE) should be displayed to the user in a special way (e.g. as buttons rather than links), making the user aware of possible side effect of their actions (e.g. financial transaction).

Methods GET, HEAD, PUT and DELETE are defined to be idempotent, meaning that multiple identical requests should have the same effect as a single request. Also, the methods OPTIONS and TRACE should not have side effects, and so are inherently idempotent.

Despite the specified idempotence of GET requests, in practice, GET requests are often used to pass HTML form values or other data to an HTTP server. These requests can cause changes on the server, through CGI execution, which may result in different effects for successive identical requests. For example, an HTML page may use a link to cause the deletion of a database record; merely GET-ing a particular URL on a server will cause the CGI application on the server to delete a record,

thus causing a change of the server's state and possibly making identical following requests to this URL to fail, on account of the database record already being deleted. This behavior is technically discouraged (non-idempotent actions should ideally be initiated by a **POST** request) but is very common on the modern World Wide Web. Such behavior can cause problems because various schemes for caching web pages, such as search engines, which by design GET pages before a user initiates a request, can cause unintentional changes on a server.

HTTP servers are supposed to implement at least GET and HEAD methods and, whenever possible, also OPTIONS method.

HTTP versions

HTTP differs from other TCP-based protocols such as FTP, because HTTP has different protocol versions:

- **0.9** Deprecated. Was never widely used. Only supports one command, GET. Does not support headers. Since this version does not support POST the client can't pass much information to the server.
- **HTTP/1.0** Still in wide use, especially by proxy servers. Allows persistent connections (alias keep-alive connections, more than one request-response per TCP/IP connection) when explicitly negotiated; however, this only works well when not using proxy servers.
- **HTTP/1.1** Current version; persistent connections enabled by default and works well with proxies. Also supports request pipelining, allowing multiple requests to be sent at the same time, allowing the server to prepare for the workload and potentially transfer the requested resources more quickly to the client.

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 "200") and a textual *reason phrase* (such as "OK"). The way the user agent handles the response primarily depends on the code and secondarily on the response headers. Custom status codes can be used since if the user agent encounters a code it does not recognize, it can use the first digit of the code to determine the general class of the response. [1]
(<https://web.archive.org/web/20060323170107/http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html#sec6.1>)

Also, the standard reason phrases are only recommendations and can be replaced with "local equivalents" at 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.

In practice, the reason phrase is unlikely to reach the user and is never interpreted. All modern web browsers rely on the status code to determine the handling and the response body to inform the user. However, the reason phrase may be logged and thus custom reason phrases might help in debugging.

See list of HTTP status codes for a list of all widely known status codes and associated standard reason phrases.

HTTP connection persistence

In HTTP/0.9 and HTTP/1.0, a client sends a request to the server, the server sends a response back to the client. After this, the connection is closed. HTTP/1.1, however, supports persistent connections. This enables the client to send a request and get a response, and then send additional requests and get additional responses. The TCP connection is not released for the multiple additional requests, so the relative overhead due to TCP is much less per request. The use of persistent connection

is often called *keep alive*. It is also possible to send more than one (usually between two and five) request before getting responses from previous requests. This is called *pipelining*.

There is a HTTP/1.0 extension for connection persistence, but its utility is limited due to HTTP/1.0's lack of unambiguous rules for delimiting messages. This extension uses a header called **Keep-Alive**, while the HTTP/1.1 connection persistence uses the **Connection** header. Therefore a HTTP/1.1 may choose to support either just HTTP/1.1 connection persistence, or both HTTP/1.0 and HTTP/1.1 connection persistence. Some HTTP/1.1 clients and servers do not implement connection persistence or have it disabled in their configuration.

HTTP connection closing

Both HTTP servers and clients are allowed to close TCP/IP connections at any time (i.e. depending on their settings, their load, etc.). This feature makes HTTP ideal for the World Wide Web, where pages regularly link to many other pages on the same server or to external servers.

Closing an HTTP/1.1 connection can be a much longer operation (from 200 milliseconds up to several seconds) than closing an HTTP/1.0 connection, because the first usually needs a **linger** close while the second can be immediately closed as soon as the entire first request has been read and the full response has been sent.

HTTP session state

HTTP can occasionally pose problems for Web developers (Web Applications), because HTTP is stateless. The advantage of a stateless protocol is that hosts don't need to retain information about users between requests, but this forces the use of alternative methods for maintaining users' state, for example when a host would like to customize content for a user who has visited before. The common method for solving this problem involves the use of sending and requesting cookies.

Secure HTTP

Main article: [https: URI scheme](#)

https: is a URI scheme syntactically identical to the **http:** scheme used for normal HTTP connections, but which signals the browser to use an added encryption layer of SSL/TLS to protect the traffic. SSL is especially suited for HTTP since it can provide some protection even if only one side to the communication is authenticated. In the case of HTTP transactions over the Internet, typically only the server side is authenticated.

Sample

Below is a sample conversation between an HTTP client and an HTTP server running on www.example.com, port 80.

Client request (followed by a blank line, so that request ends with a double newline, each in the form of a carriage return followed by a line feed):

```
GET /index.html HTTP/1.1
Host: www.example.com
```

The "Host" header 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.

Server response (followed by a blank line and text of the requested page):

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.27 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

See also

- 404 error
- Uniform resource locator
- Basic authentication scheme
- Digest access authentication
- Captive portal
- HTTP proxy
- Content negotiation
- WebDAV

External links

Specifications and references

- HTTP/1.0 specification (May 1996) as plain text: RFC 1945
- HTTP/1.1 specification (June 1999) as plain text: RFC 2616; also as HTML (<https://web.archive.org/web/20060323170107/http://www.w3.org/Protocols/rfc2616/rfc2616.html>) , as PostScript (<https://web.archive.org/web/20060323170107/ftp://ftp.isi.edu/in-notes/rfc2616.ps>) , and as PDF (<https://web.archive.org/web/20060323170107/http://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf>) ;
- HTTP/1.1 specification errata (<https://web.archive.org/web/20060323170107/http://purl.org/NET/http-errata>)
- Tim Berners-Lee's original 1992 Internet-Draft (<https://web.archive.org/web/20060323170107/http://www.w3.org/Protocols/HTTP/HTTP2.html>)
- HTTP Sequence Diagram (https://web.archive.org/web/20060323170107/http://www.eventhelix.com/RealtimeMantra/Networking/http_sequence_diagram.pdf) (PDF)

Tutorials and tools

- HTTP Made Really Easy (<https://web.archive.org/web/20060323170107/http://www.jmarshall.com/easy/http/>)
- HTTP header viewer (<https://web.archive.org/web/20060323170107/http://analyze.forret.com/>)
- HTTP Header Check - Bookmarklet (<https://web.archive.org/web/20060323170107/http://www.webconfs.com/http-header-check.php>)
- View HTTP Request and Response Header (<https://web.archive.org/web/20060323170107/http://web-sniffer.net/>)
- Command-line HTTP clients: cURL (<https://web.archive.org/web/20060323170107/http://curl.haxx.se/>) , Wget (<https://web.archive.org/web/20060323170107/http://www.gnu.org/software/wget/wget.html>) , Snarf (<https://web.archive.org/web/20060323170107/http://www.xach.com/snarf/>) , fetch (<https://web.archive.org/web/20060323170107/http://www.freebsd.org/cgi/cvsweb.cgi/src/usr.bin/fetch/>)
- HTTP Request Smuggling (<https://web.archive.org/web/20060323170107/http://www.watchfire.com/resources/HTTP-Request-Smuggling.pdf>) (PDF) (registration required)
- HTTP compression (<https://web.archive.org/web/20060323170107/http://www.http-compression.com/>)
- Live HTTP Headers Extension for Firefox (<https://web.archive.org/web/20060323170107/http://livehttpheaders.mozdev.org/>)

Retrieved from "http://en.wikipedia.org/wiki/HyperText_Transfer_Protocol"

Categories: Internet protocols | Internet standards | HTTP

- This page was last modified 08:48, 21 March 2006.
- All text is available under the terms of the GNU Free Documentation License (see **Copyrights** for details).
Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc.
- Privacy policy
- About Wikipedia
- Disclaimers