The Wayback Machine - https://web.archive.org/web/20090329013006/http://en.wikipedia.org:80/wiki/HTTP

Hypertext Transfer Protocol

From Wikipedia, the free encyclopedia
(Redirected from HTTP)

Hypertext Transfer Protocol (**HTTP**) is an application-level protocol for distributed, collaborative, hypermedia information systems.[1] Its use for retrieving inter-linked resources led to the establishment of the World Wide Web.

HTTP development was coordinated by the World Wide Web Consortium and the Internet Engineering Task Force (IETF), culminating in the publication of a series of Requests for Comments (RFCs), most notably RFC 2616 (June 1999), which defines HTTP/1.1, the version of HTTP in common use.

HTTP is a request/response standard of a client and a server. A client is the end-user, the server is the web site. The client making a HTTP request—using a web browser, spider, or other end-user tool—is referred to as the *user agent*. The responding server—which stores or creates *resources* such as HTML files and images—is called the *origin server*. In between the user agent and origin server may be several intermediaries, such as proxies, gateways, and tunnels. HTTP is not constrained to using TCP/IP and its supporting layers, although this is its most popular application on the Internet. Indeed HTTP can be "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."[2]

Typically, an HTTP client initiates a request. It establishes a Transmission Control Protocol (TCP) connection to a particular port on a host (port 80 by default; see List of TCP and UDP port numbers). An 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 resource, an error message, or some other information.

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

Contents

- 1 Request message
  2 Request methods
  - 2.1 Safe methods
  - 2.2 Idempotent methods and web applications
- 3 Status codes
- 4 Persistent connections
- 5 HTTP session state
  6 Secure HTTP
  - 6.1 HTTPS URI scheme
  - 6.2 HTTP 1.1 Upgrade header
- 7 Sample
- 8 See also
- 9 References
- 10 External links

The Internet Protocol Suite

Application Layer

BGP · DHCP · DNS · FTP · GTP · **HTTP** · IMAP · IRC · Megaco · MGCP · NNTP · NTP · POP · RIP · RPC · RTP · RTSP · SDP · SIP · SMTP · SNMP · SOAP · SSH · Telnet · TLS/SSL · XMPP · (more)

Transport Layer

TCP · UDP · DCCP · SCTP · RSVP · ECN · (more)

Internet Layer

IP (IPv4, IPv6) · ICMP · ICMPv6 · IGMP · IPsec · (more)

Dropbox Exhibit 1027 - Page 1
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

Request message

Link Layer

ARP · RARP · NDP · OSPF · Tunnels (L2TP) · Media Access Control (Ethernet, MPLS, DSL, ISDN, FDDI) · Device Drivers · (more)

The request message consists of the following:

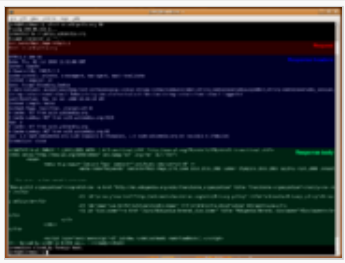
- Request line, such as `GET /images/logo.gif HTTP/1.1`, which requests a resource called `/images/logo.gif` from server
- Headers, such as `Accept-Language: en`
- An empty line
- An optional message body

The request line and headers must all end with <CR><LF> (that is, a carriage return followed by a line feed). The empty line must consist of only <CR><LF> and no other whitespace. In the HTTP/1.1 protocol, all headers except Host 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.

Request methods

HTTP defines eight methods (sometimes referred to as "verbs") indicating the desired action to be performed on the identified **resource**. What this resource represents, whether pre-existing 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.



A HTTP request made using telnet. The request, response headers and response body are highlighted.

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.

GET
: Requests a representation of the specified resource. Note that GET should not be used for operations that cause side-effects, such as using it for taking actions in web applications. One reason for this is that GET may be used arbitrarily by robots or crawlers, which should not need to consider the side effects that a request should cause. See safe methods below.

POST
: Submits data to be processed (e.g., from an HTML form) to the identified resource. The data is included in the body of the request. This may result in the creation of a new resource or the updates of existing resources or both.

PUT
: Uploads a representation of the specified resource.

DELETE
: Deletes the specified resource.

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 for specified URL. This can be used to check the functionality of a web server by requesting '*' instead of a specific resource.

CONNECT

Dropbox Exhibit 1027 - Page 2
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

Converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communication (HTTPS) through an unencrypted HTTP proxy.[3]

HTTP servers are required to implement at least the GET and HEAD methods[4] and, whenever possible, also the OPTIONS method.

Safe methods

Some methods (for example, HEAD, GET, OPTIONS and TRACE) are defined as *safe*, which means they are intended only for information retrieval and should not change the state of the server. In other words, they should not have side effects, beyond relatively harmless effects such as logging, caching, the serving of banner advertisements or incrementing a web counter. Making arbitrary GET requests without regard to the context of the application's state should therefore be considered safe.

By contrast, methods such as POST, PUT and DELETE are intended for actions which may cause side effects either on the server, or external side effects such as financial transactions or transmission of email. Such methods are therefore not usually used by conforming web robots or web crawlers, which 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, and careless or deliberate programming can just as easily (or more easily, due to lack of user agent precautions) cause non-trivial changes on the server. This is discouraged, because it can cause problems for Web caching, search engines and other automated agents, which can make unintended changes on the server.

Idempotent methods and web applications

Methods PUT and DELETE are defined to be idempotent, meaning that multiple identical requests should have the same effect as a single request. Methods GET, HEAD, OPTIONS and TRACE, being prescribed as safe, should also be idempotent, as HTTP is a stateless protocol.

By contrast, the POST method is not necessarily idempotent, and therefore sending an identical POST request multiple times may further affect state or cause further side effects (such as financial transactions). In some cases this may be desirable, but in other cases this could be due to an accident, such as when a user does not realize that their action will result in sending another request, or they did not receive adequate feedback that their first request was successful. 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 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. Ignoring this recommendation, however, may result in undesirable consequences if a user agent assumes that repeating the same request is safe when it isn't.

Status codes

See also: List of HTTP 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 way the user agent handles the response primarily depends on the code and secondarily on the response headers. Custom status codes can be used

Dropbox Exhibit 1027 - Page 3
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

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.[5]

Also, 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.

Persistent connections

In HTTP/0.9 and 1.0, the connection is closed after a single request/response pair. In HTTP/1.1 a keep-alive-mechanism was introduced, where a connection could be reused for more than one request.

Such *persistent connections* reduce lag perceptibly, because the client does not need to re-negotiate the TCP connection after the first request has been sent.

Version 1.1 of the protocol made bandwidth optimization improvements to HTTP/1.0. For example, HTTP/1.1 introduced chunked transfer encoding to allow content on persistent connections to be streamed, rather than buffered. HTTP pipelining further reduces lag time, allowing clients to send multiple requests before a previous response has been received to the first one. Another improvement to the protocol was byte serving, which is when a server transmits just the portion of a resource explicitly requested by a client.

HTTP session state

HTTP is a stateless protocol. The advantage of a stateless protocol is that hosts do not need to retain information about users between requests, but this forces web developers to use alternative methods for maintaining users' states. For example, when a host needs to customize the content of a website for a user, the web application must be written to track the user's progress from page to page. A common method for solving this problem involves sending and receiving cookies. Other methods include server side sessions, hidden variables (when the current page is a form), and URL encoded parameters (such as `/index.php?session_id=some_unique_session_code`).

Secure HTTP

There are currently two methods of establishing a secure HTTP connection: the HTTPS URI scheme and the HTTP 1.1 `Upgrade` header, introduced by RFC 2817. Browser support for the `Upgrade` header is, however, nearly non-existent, hence the HTTPS URI scheme is still the dominant method of establishing a secure HTTP connection. Secure HTTP is notated by the prefix HTTPS:// instead of HTTP://

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 of the communication is authenticated. This is the case with HTTP transactions over the Internet, where typically only the server is authenticated (by the client examining the server's certificate).

HTTP 1.1 Upgrade header

Dropbox Exhibit 1027 - Page 4
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

HTTP 1.1 introduced support for the `Upgrade` header. In the exchange, the client begins by making a clear-text request, which is later upgraded to TLS. Either the client or the server may request (or demand) that the connection be upgraded. The most common usage is a clear-text request by the client followed by a server demand to upgrade the connection, which looks like this:

Client:

```
GET /encrypted-area HTTP/1.1
Host: www.example.com
```

Server:

```
HTTP/1.1 426 Upgrade Required
Upgrade: TLS/1.0, HTTP/1.1
Connection: Upgrade
```

The server returns a 426 status-code because 400 level codes indicate a client failure (see List of HTTP status codes), which correctly alerts legacy clients that the failure was client-related.

The benefits of using this method for establishing a secure connection are:

- that it removes messy and problematic redirection and URL rewriting on the server side,
- it allows virtual hosting of secured websites (although HTTPS also allows this using Server Name Indication), and
- it reduces user confusion by providing a single way to access a particular resource.

A weakness with this method is that the requirement for a secure HTTP cannot be specified in the URI. In practice, the (untrusted) server will thus be responsible for enabling secure HTTP, not the (trusted) client.

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/1.3.3.7 (Unix)  (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
```


Dropbox Exhibit 1027 - Page 5
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

```
Connection: close
Content-Type: text/html; charset=UTF-8
```

The ETag (entity tag) header 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 header **Accept-Ranges: bytes**. This is useful if the client needs to have only certain portions[6] of a resource sent by the server, which is called byte serving. When **Connection: close** is sent in a header, it means that the web server will close the TCP connection immediately after the transfer of this package.

See also

- Basic access authentication
- Content negotiation
- Digest access authentication
- HTTP compression
- List of file transfer protocols
- List of HTTP status codes
- List of HTTP headers
- WebDAV
- Web cache

References

1. ^ http://www.ietf.org/internet-drafts/draft-ietf-httpbis-p1-messaging-05.txt
2. ^ Fielding, et al. "Internet RFC 2616." (https://web.archive.org/web/20090329013006/http://www.w3.org/Protocols/rfc2616/rfc2616-sec1.html#sec1.4) , section 1.4. Retrieved on January 21, 2009.
3. ^ "Vulnerability Note VU#150227: HTTP proxy default configurations allow arbitrary TCP connections". US-CERT. 2002-05-17. http://www.kb.cert.org/vuls/id/150227. Retrieved on 2007-05-10.
4. ^ HTTP 1.1 Section 5.1.1 (https://web.archive.org/web/20090329013006/http://tools.ietf.org/html/rfc2616#section-5.1.1)
5. ^ 6.1 Status-Line (https://web.archive.org/web/20090329013006/http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html#sec6.1)
6. ^ draft-ietf-http-range-retrieval-00 - Byte Range Retrieval Extension to HTTP (https://web.archive.org/web/20090329013006/http://tools.ietf.org/html/draft-ietf-http-range-retrieval-00)

External links

- Bulkcheck HTTP-Headers from different URLs simultaneously (https://web.archive.org/web/20090329013006/http://pc-intern.com/http-header-bulkcheck.html)
- What GET requests sent by browsers look like (https://web.archive.org/web/20090329013006/http://mprzybylski.pl/lab/net/http/browser-requests.html)
- Watch HTTP Client/Server Request/Responses (https://web.archive.org/web/20090329013006/http://www.httpviewer.net/)

Retrieved from "http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol"
Categories: Semantic Web | HTTP | Network protocols | Web browsers | Internet protocols | Application layer protocols | World Wide Web | Open formats | World Wide Web Consortium standards
Hidden categories: All articles with unsourced statements | Articles with unsourced statements since August 2008

- This page was last modified on 27 March 2009, at 06:43.

Dropbox Exhibit 1027 - Page 6
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338

- 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., a U.S. registered 501(c)(3) tax-deductible nonprofit charity.

Dropbox Exhibit 1027 - Page 7
Dropbox, Inc. v. Entangled Media, LLC
IPR2024-00284 - U.S. Patent No. 8,296,338