Hypertext Transfer Protocol

Juha Söderqvist

Hypertext Transfer Protocol

HTTP is the protocol to exchange or transfer hypertext

Development of HTTP was initiated by Tim Berners-Lee at CERN in 1989. Standards development of HTTP was coordinated by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C), culminating in the publication of a series of Requests for Comments (RFCs). The first definition of HTTP/1.1, the version of HTTP in common use, occurred in RFC 2068 in 1997

HTTP/2, was standardized in 2015, and is now supported by major web servers.

Hypertext Transfer Protocol

HTTP functions as a request–response protocol in the client–server computing model. A web browser, for example, may be the client and an application running on a computer hosting a website 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.

HTTP – User Agent

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 request headers, the information in the "User-Agent" string contributes to the information that the client sends to the server, since the string can vary considerably from user to user

HTTP – User Agent

During the first browser war, many web servers were configured to only send web pages that required advanced features, including frames, to clients that were identified as some version of Mozilla. Other browsers were considered to be older products such as Mosaic, Cello or Samba and would be sent a bare bones HTML document.

For this reason, most Web browsers use a User-Agent string value as follows:

Mozilla/[version] ([system and browser information]) [platform] ([platform details]) [extensions].

HTTP – User Agent

Safari on the iPad has used the following:

Mozilla/5.0 (iPad; U; CPU OS 3_2_1 like Mac OS X; en-us) AppleWebKit/531.21.10 (KHTML, like Gecko) Mobile/7B405

- The components of this string are as follows:
- Mozilla/5.0: Previously used to indicate compatibility with the Mozilla rendering engine.
- (iPad; U; CPU OS 3_2_1 like Mac OS X; en-us): Details of the system in which the browser is running.
- AppleWebKit/531.21.10: The platform the browser uses.
- (KHTML, like Gecko): Browser platform details.
- Mobile/7B405: This is used by the browser to indicate specific enhancements that are available directly in the browser or through third parties

HTTP – User Agent

Opera was the most widely used web browser that did not have the User-Agent string with "Mozilla" (instead beginning it with "Opera"). Since July 15, 2013, Opera's User-Agent string begins with "Mozilla/5.0" and, to avoid encountering legacy server rules, no longer includes the word "Opera" (instead using the string "OPR" to denote the Opera version)

HTTP – User Agent

Automated web crawling tools can use a simplified form, where an important field is contact information in case of problems. By convention the word "bot" is included in the name of the agent.

Googlebot/2.1
(+http://www.google.com/bot.html)

Automated agents are expected to follow rules in a special file called "robots.txt".

HTTP – User Agent spoofing

The popularity of various Web browser products has varied throughout the Web's history, and this has influenced the design of Web sites in such a way that Web sites are sometimes designed to work well only with particular browsers, rather than according to uniform standards by the World Wide Web Consortium (W3C) or the Internet Engineering Task Force (IETF)

HTTP – User Agent detection

Web sites often include code to detect browser version to adjust the page design sent according to the user agent string received. This may mean that less-popular browsers are not sent complex content (even though they might be able to deal with it correctly) or, in extreme cases, refused all content. Thus, various browsers have a feature to cloak or spoof their identification to force certain server-side content. For example, the Android browser identifies itself as Safari (among other things) in order to aid compatibility.

HTTP – User Agent string

Other HTTP client programs, like download managers and offline browsers, often have the ability to change the user agent string.

Spam bots and Web scrapers often use fake user agents.

At times it has been popular among Web developers to initiate Viewable With Any Browser campaigns,[10] encouraging developers to design Web pages that work equally well with any browser.

A result of user agent spoofing may be that collected statistics of Web browser usage are inaccurate

HTTP – User Agent sniffing

The term user agent sniffing refers to the practice of Web sites showing different content when viewed with a certain user agent. On the Internet, this will result in a different site being shown when browsing the page with a specific browser.

HTTP – User Agent sniffing

The term user agent sniffing refers to the practice of Web sites showing different content when viewed with a certain user agent. On the Internet, this will result in a different site being shown when browsing the page with a specific browser.

When viewed with Internet Explorer 6 or newer, more functionality is displayed compared to the same page in any other browsers, because other browsers may not be able to render content specifically designed for Internet Explorer 6. User agent sniffing is now considered poor practice, since it encourages browser-specific design and penalizes new browsers with unrecognized user agent identifications. Instead, the W3C recommends creating HTML markup that is standard, allowing correct rendering in as many browsers as possible, and to test for specific browser features rather than particular browser versions or brands

HTTP – User Agent sniffing

Web sites specifically targeted towards mobile phones, like NTT DoCoMo's I-Mode or Vodafone's Vodafone Live! portals, often rely heavily on user agent sniffing, since mobile browsers often differ greatly from each other. Many developments in mobile browsing have been made in the last few years

HTTP – Cache

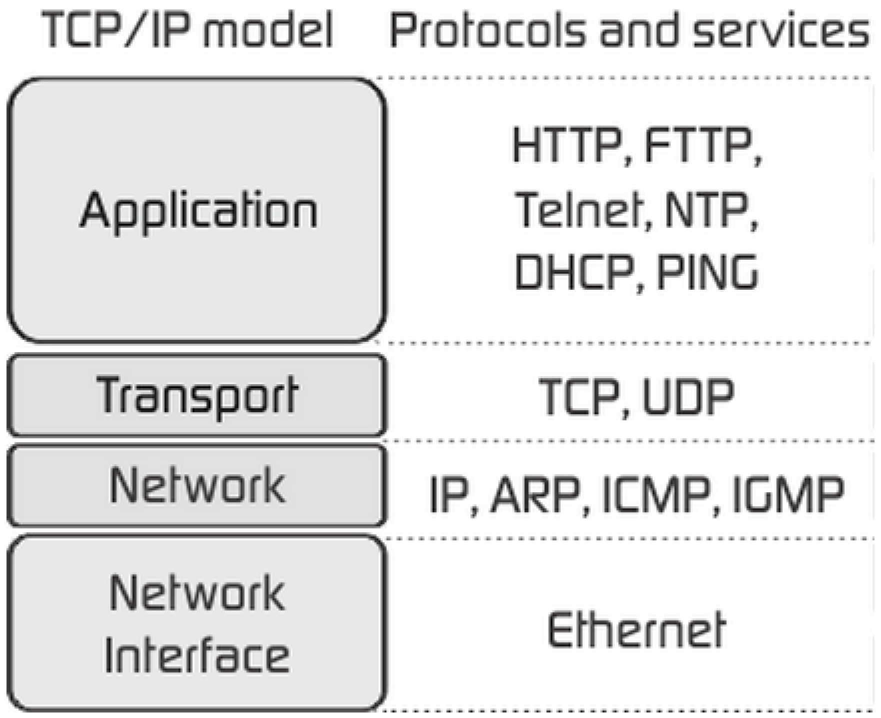
HTTP is designed to permit intermediate network elements to improve or enable communications between clients and servers. High-traffic websites often benefit from web cache servers that deliver content on behalf of upstream servers to improve response time.

Web browsers cache previously accessed web resources and reuse them when 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.

HTTP – Application Layer

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, and Transmission Control Protocol (TCP) is commonly used.



HTTP – Session

An HTTP session is a sequence of network request-response transactions. An HTTP client initiates a request by establishing a Transmission Control Protocol (TCP) connection to a particular port on a server (typically port 80, occasionally port 8080

An HTTP server listening on that port waits for a client's 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 this message is typically the requested resource, although an error message or other information may also be returned

HTTP – Port

The port numbers in the range from 0 to 1023 are the well-known ports or system ports

- 25 SMTP simple mail transfer protocol
- 53 DNS domain name system
- 80 Hypertext transfer protocol HTTP
- 443 Hypertext transfer protocol HTTP over TLS/SSL   (HTTPS)
- 445 Microsoft-DS Active Directory, Windows shares

HTTP Authentication

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

HTTP Basic authentication (BA) implementation is the simplest technique for enforcing access controls to web resources because it doesn't require cookies, session identifiers, or login pages; rather, HTTP Basic authentication uses standard fields in the HTTP header, obviating the need for handshakes.

HTTP Authentication

When the server wants the user agent to authenticate itself towards the server, it must respond appropriately to unauthenticated requests.

Unauthenticated requests should return a response whose header contains a HTTP 401 Unauthorized status

HTTP Authentication

Digest access authentication is one of the agreed-upon methods a web server can use to negotiate credentials, such as username or password, with a user's web browser. This can be used to confirm the identity of a user before sending sensitive information, such as online banking transaction history. It applies a hash function to the username and password before sending them over the network. In contrast, basic access authentication uses the easily reversible Base64 encoding instead of encryption, making it non-secure unless used in conjunction with SSL.

HTTP Authentication

Client request (no authentication)

```
GET /dir/index.html HTTP/1.0
Host: server
```

Server response

```
HTTP/1.0 401 Unauthorized
Server: HTTPd/0.9
Date: Sun, 10 Apr 2014 20:26:47 GMT
WWW-Authenticate: Digest realm="testrealm@host.com",
                        qop="auth,auth-int",
                        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                        opaque="5ccc069c403ebaf9f0171e9517f40e41"
Content-Type: text/html
Content-Length: 153

<!DOCTYPE html>
<html>
  <head>
   <meta charset="UTF-8" />
   <title>Error</title>  </head>  <body>
   <h1>401 Unauthorized.</h1>
</body></html>
```

HTTP Authentication - MD5

- String: Juha:testrealm@host.com:Circle Of Life
- MD5 HASH String
- 9a077c314567c02ee054940ac63eeb18

  The MD5 algorithm is a widely used hash function producing a 128-bit hash value
- cryptographic hash function is a special class of hash function that has certain properties which make it suitable for use in cryptography. It is a mathematical algorithm that maps data of arbitrary size to a bit string of a fixed size (a hash function) which is designed to also be a one-way function, that is, a function which is infeasible to invert

HTTP Authentication - MD5

cryptographic hash function has five main properties:

- it is deterministic so the same message always results in the same hash
- it is quick to compute the hash value for any given message
- it is infeasible to generate a message from its hash value except by trying all possible messages
- a small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value
- it is infeasible to find two different messages with the same hash value

HTTP Authentication - Juha

Client request (username "Juha", password "Circle Of Life"

```
GET /dir/index.html HTTP/1.0
Host: localhost
Authorization: Digest username="Juha",
               realm="testrealm@host.com",
                nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                uri="/dir/index.html",
                qop=auth,
                nc=00000001,
                cnonce="0a4f113b",
                response="6629fae49393a05397450978507c4ef1",
                opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

HTTP Authentication - Juha

Server response

```
HTTP/1.0 200 OK
Server: HTTPd/0.9
Date: Sun, 10 Apr 2005 20:27:03 GMT
Content-Type: text/html
Content-Length: 7984
```

HTTP Authentication - Juha

The "response" value is calculated in three steps, as follows. Where values are combined, they are delimited by colons.

1. The MD5 hash of the combined username, authentication realm and password is calculated. The result is referred to as HA1.
2. The MD5 hash of the combined method and digest URI is calculated, e.g. of "GET" and "/dir/index.html". The result is referred to as HA2.
3. The MD5 hash of the combined HA1 result, server nonce (nonce), request counter (nc), client nonce (cnonce), quality of protection code (qop) and HA2 result is calculated. The result is the "response" value provided by the client.

Since the server has the same information as the client, the response can be checked by performing the same calculation. In the example given above the result is formed as follows, where MD5() represents a function used to calculate an MD5 hash, backslashes represent a continuation and the quotes shown are not used in the calculation.

HTTP Authentication - Juha

HA1 = MD5("Juha:testrealm@host.com:Circle Of Life")

- = 9a077c314567c02ee054940ac63eeb18
-
- HA2 = MD5("GET:/dir/index.html")
- = 39aff3a2bab6126f332b942af96d3366
-
- Response = MD5("9a077c314567c02ee054940ac63eeb18:\
- dcd98b7102dd2f0e8b11d0f600bfb0c093:\
- 00000001:0a4f113b:auth:\
- 39aff3a2bab6126f332b942af96d3366")
- = 6629fae49393a05397450978507c4ef1

HTTP Request methods

HTTP defines methods (sometimes referred to as verbs) to indicate 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

HTTP Request methods

GET

The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect.

HEAD

- The HEAD method asks for a response identical to that of 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.

HTTP Request methods

POST

- The POST method requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, for example, an annotation for existing resources; a message for a bulletin board, newsgroup, mailing list, or comment thread; a block of data that is the result of submitting a web form to a data-handling process; or an item to add to a database.

HTTP Request methods

Method/Description

- HEAD Same as GET but returns only HTTP headers and no document body
- PUT Uploads a representation of the specified URI
- DELETE Deletes the specified resource
- OPTIONS   Returns the HTTP methods that the server supports
- CONNECT Converts the request connection to a transparent TCP/IP tunnel

HTTP Request methods

Two HTTP Request Methods: GET and POST

Two commonly used methods for a request-response between a client and server are: GET and POST.

- GET - Requests data from a specified resource
- POST - Submits data to be processed to a specified resource

HTTP Request methods - GET

/test/demo_form.asp?
name1=value1&name2=value2

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests should be used only to retrieve data

HTTP Request methods - POST

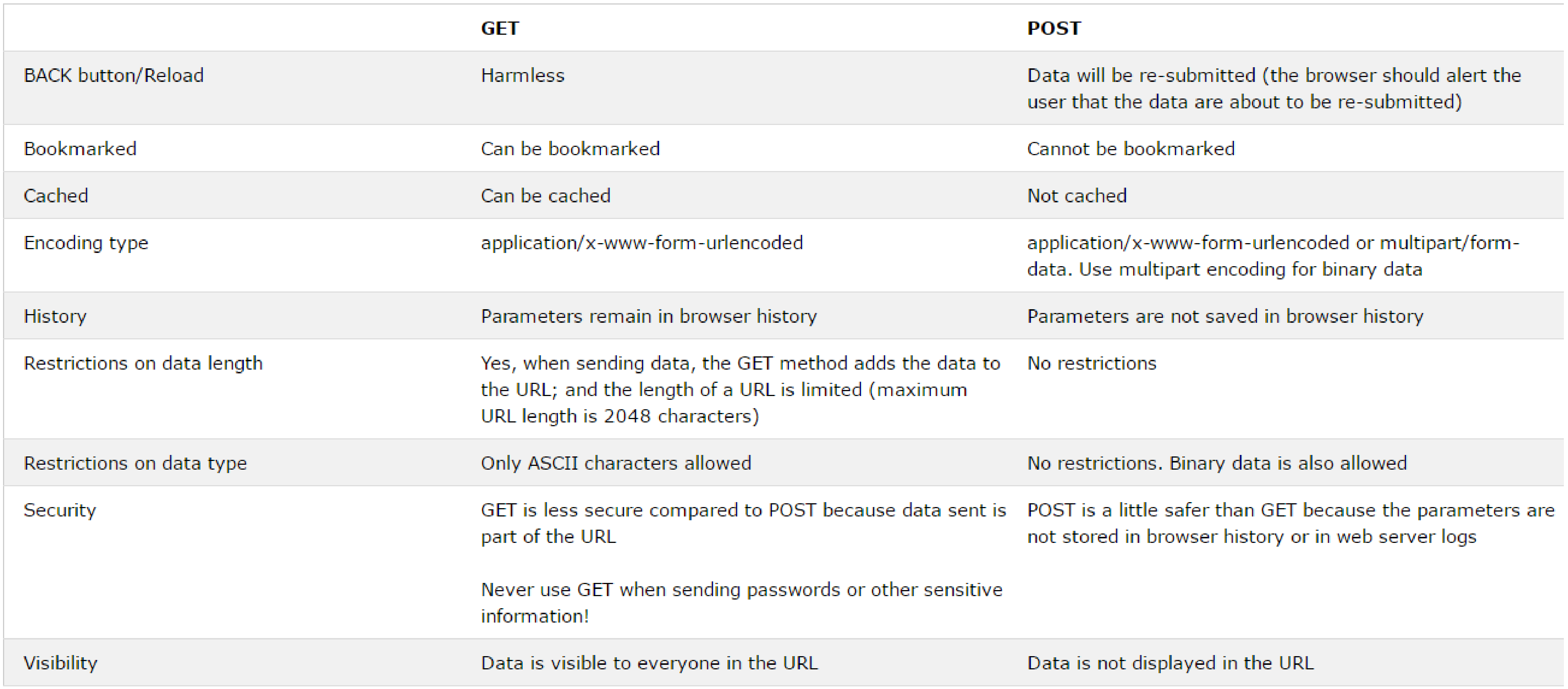
The POST Method

- Note that the query string (name/value pairs) is sent in the HTTP message body of a POST request:

```
POST /test/demo_form.asp HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
```

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

HTTP – GET/POST



	GET	**POST**
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

HTTP Status Messages

1xx: Information

- 100 Continue The server has received the request headers, and the client should proceed to send the request body
- 101 Switching Protocols The requester has asked the server to switch protocols
- 103 Checkpoint Used in the resumable requests proposal to resume aborted PUT or POST requests
- 2xx: Successful
- 200 OK The request is OK (this is the standard response for successful HTTP requests)
- 201 Created The request has been fulfilled, and a new resource is created
- 202 Accepted The request has been accepted for processing, but the processing has not been completed
- 203 Non-Authoritative Information The request has been successfully processed, but is returning information that may be from another source
- 204 No Content The request has been successfully processed, but is not returning any content
- 205 Reset Content The request has been successfully processed, but is not returning any content, and requires that the requester reset the document view
- 206 Partial Content The server is delivering only part of the resource due to a range header sent by the client

HTTP Status Messages

3xx: Redirection

- 300 Multiple Choices A link list. The user can select a link and go to that location. Maximum five addresses
- 301 Moved Permanently The requested page has moved to a new URL
- 302 Found The requested page has moved temporarily to a new URL
- 303 See Other The requested page can be found under a different URL
- 304 Not Modified Indicates the requested page has not been modified since last requested
- 306 Switch Proxy No longer used
- 307 Temporary Redirect The requested page has moved temporarily to a new URL
- 308 Resume Incomplete Used in the resumable requests proposal to resume aborted PUT or POST requests

HTTP Status Messages

4xx: Client Error

- 400 Bad Request The request cannot be fulfilled due to bad syntax
- 401 Unauthorized The request was a legal request, but the server is refusing to respond to it. For use when authentication is possible but has failed or not yet been provided
- 402 Payment Required Reserved for future use
- 403 Forbidden The request was a legal request, but the server is refusing to respond to it
- 404 Not Found The requested page could not be found but may be available again in the future
- 405 Method Not Allowed A request was made of a page using a request method not supported by that page
- 406 Not Acceptable The server can only generate a response that is not accepted by the client
- 407 Proxy Authentication Required The client must first authenticate itself with the proxy
- 408 Request Timeout The server timed out waiting for the request
- 409 Conflict The request could not be completed because of a conflict in the request
- 410 Gone The requested page is no longer available
- 411 Length Required The "Content-Length" is not defined. The server will not accept the request without it
- 412 Precondition Failed The precondition given in the request evaluated to false by the server
- 413 Request Entity Too Large The server will not accept the request, because the request entity is too large
- 414 Request-URI Too Long The server will not accept the request, because the URL is too long. Occurs when you convert a POST request to a GET request with a long query information
- 415 Unsupported Media Type The server will not accept the request, because the media type is not supported
- 416 Requested Range Not Satisfiable The client has asked for a portion of the file, but the server cannot supply that portion
- 417 Expectation Failed The server cannot meet the requirements of the Expect request-header field

HTTP Status Messages

5xx: Server Error

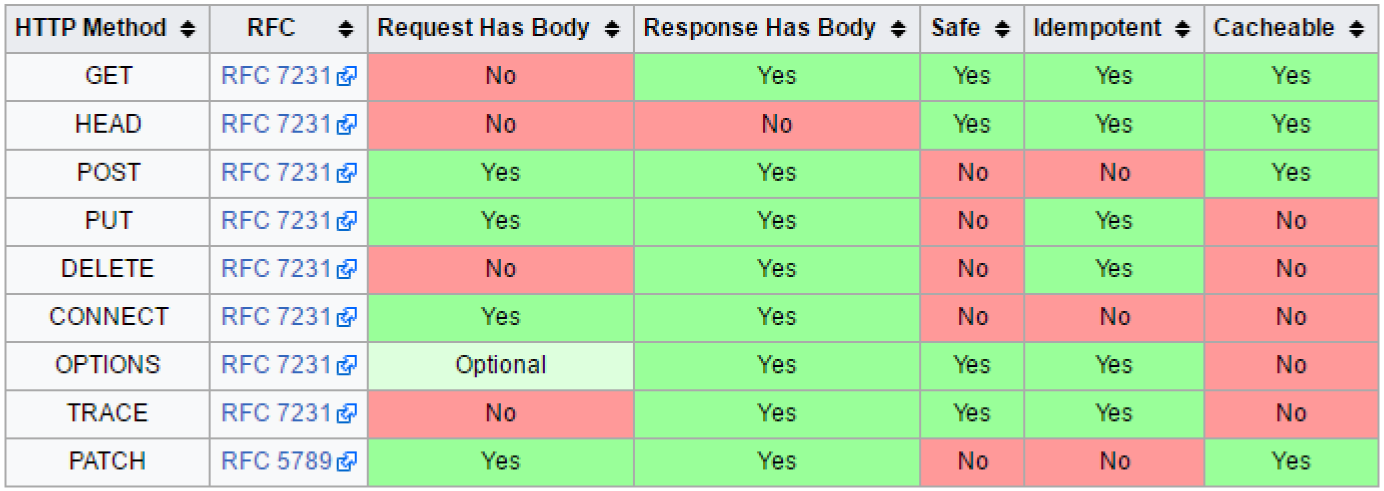
- 500 Internal Server Error A generic error message, given when no more specific message is suitable
- 501 Not Implemented The server either does not recognize the request method, or it lacks the ability to fulfill the request
- 502 Bad Gateway The server was acting as a gateway or proxy and received an invalid response from the upstream server
- 503 Service Unavailable The server is currently unavailable (overloaded or down)
- 504 Gateway Timeout The server was acting as a gateway or proxy and did not receive a timely response from the upstream server
- 505 HTTP Version Not Supported The server does not support the HTTP protocol version used in the request
- 511 Network Authentication Required The client needs to authenticate to gain network access

HTTP Request methods

HEAD, GET, OPTIONS and TRACE are, by convention, defined as safe, which means they are intended only for information retrieval and should not change the state of the server

POST, PUT, DELETE and PATCH are intended for actions that may cause side effects either on the server, or external side effects such as financial transactions or transmission of email.

HTTP methods



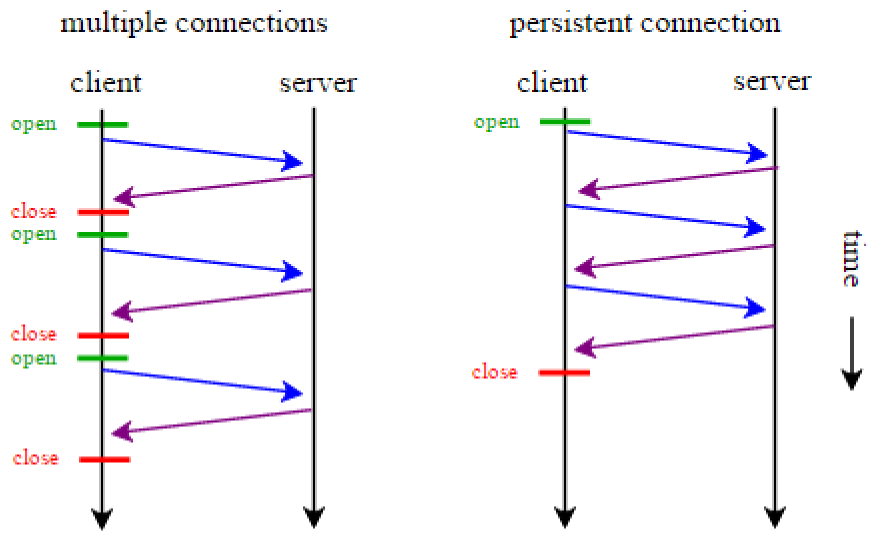
HTTP Method	RFC	Request Has Body	Response Has Body	Safe	Idempotent	Cacheable
GET	RFC 7231	No	Yes	Yes	Yes	Yes
HEAD	RFC 7231	No	No	Yes	Yes	Yes
POST	RFC 7231	Yes	Yes	No	No	Yes
PUT	RFC 7231	Yes	Yes	No	Yes	No
DELETE	RFC 7231	No	Yes	No	Yes	No
CONNECT	RFC 7231	Yes	Yes	No	No	No
OPTIONS	RFC 7231	Optional	Yes	Yes	Yes	No
TRACE	RFC 7231	No	Yes	Yes	Yes	No
PATCH	RFC 5789	Yes	Yes	No	No	Yes

HTTP Persistent connections

HTTP/1.1 a keep-alive-mechanism was introduced, where a connection could be reused for more than one request. Such persistent connections reduce request latency

HTTP persistent connections do not use separate keepalive messages, they just allow multiple requests to use a single connection. However, the default connection timeout of Apache httpd 1.3 and 2.0 is as little as 15 seconds and just 5 seconds for Apache httpd 2.2 and above. The advantage of a short timeout is the ability to deliver multiple components of a web page quickly while not consuming resources to run multiple server processes or threads for too long.

HTTP Persistent connections



HTTP session state

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

But some web applications may have to track the user's progress from page to page, for example when a web server is required to customize the content of a web page for a user.

HTTP has persistent connections (Keep-Alive), several requests can be send in same TCP Connection

Solutions for these cases include:

- the use of HTTP cookies.
- server side sessions,
- hidden variables (when the current page contains a form), and
- URL-rewriting using URI-encoded parameters, e.g., /index.php?session_id=some_unique_session_code.

HTTP Message Format

Response message

- The response message consists of the following:

A status line which includes the status code and reason message (e.g., HTTP/1.1 200 OK, which indicates that the client's request succeeded).

Response header fields (e.g., Content-Type: text/html).

An empty line.

An optional message body.

HTTP Message Format

Response message

- The response message consists of the following:

A status line which includes the status code and reason message (e.g., HTTP/1.1 200 OK, which indicates that the client's request succeeded).

Response header fields (e.g., Content-Type: text/html).

An empty line.

An optional message body.

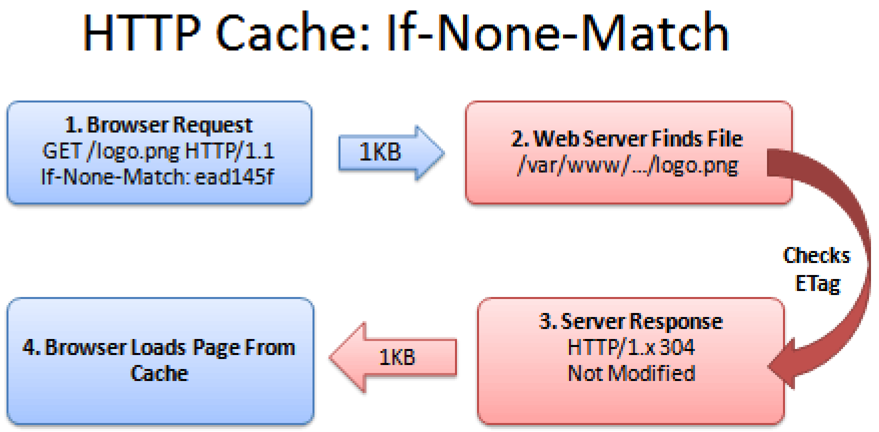
HTTP Message Format

Server response

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
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>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

HTTP ETag

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



HTTP Uniform Resource Locator

A URL (Uniform Resource Locator) is used to uniquely identify a resource over the web. URL has the following syntax:

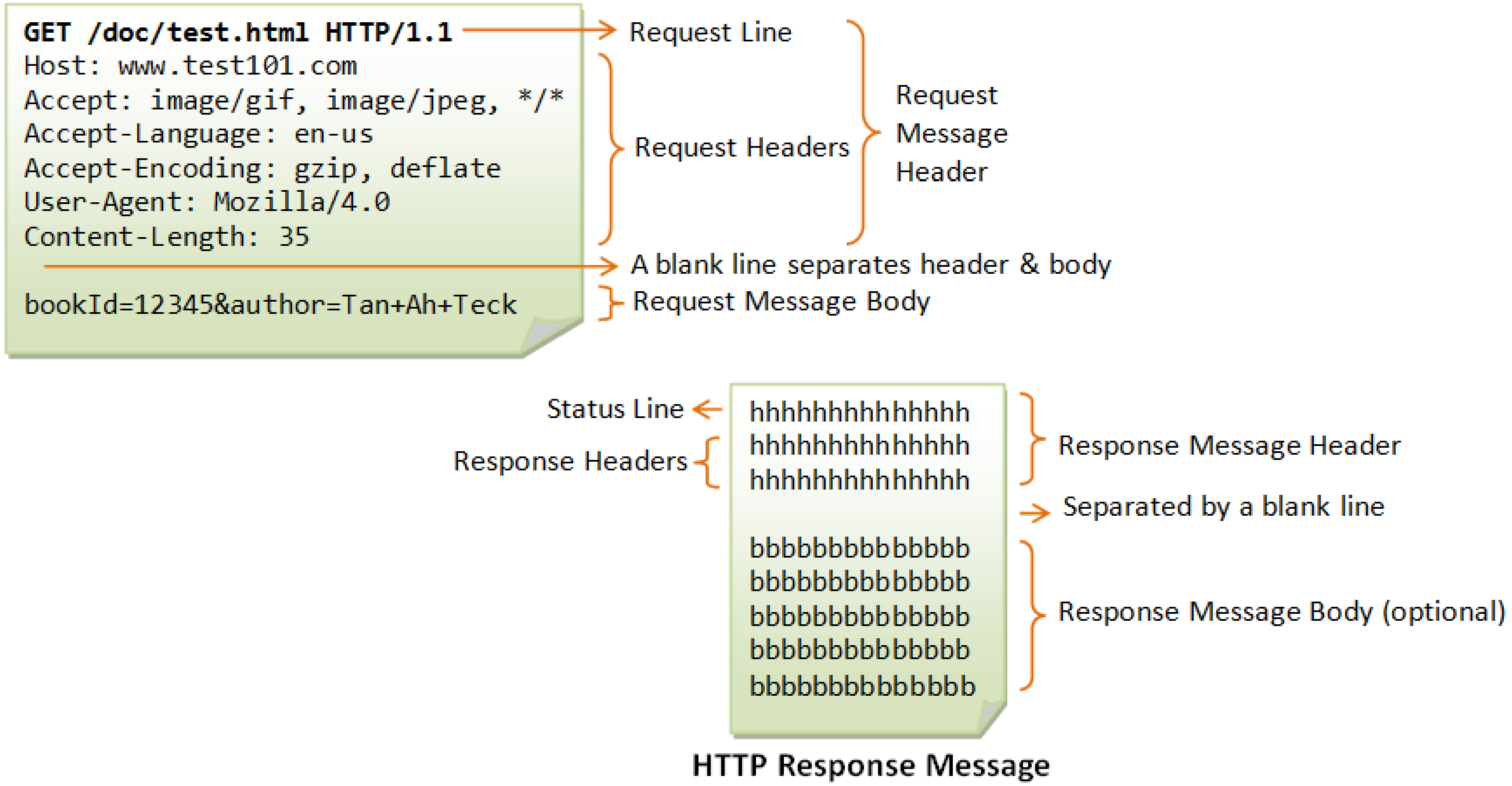
protocol://hostname:port/path-and-file-name

There are 4 parts in a URL:

- Protocol: The application-level protocol used by the client and server, e.g., HTTP, FTP, and telnet.
- Hostname: The DNS domain name (e.g., www.nowhere123.com) or IP address (e.g., 192.128.1.2) of the server.
- Port: The TCP port number that the server is listening for incoming requests from the clients.
- Path-and-file-name: The name and location of the requested resource, under the server document base directory.

HTTP Client Server Messages

HTTP client and server communicate by sending text messages. The client sends a request message to the server. The server, in turn, returns a response message.

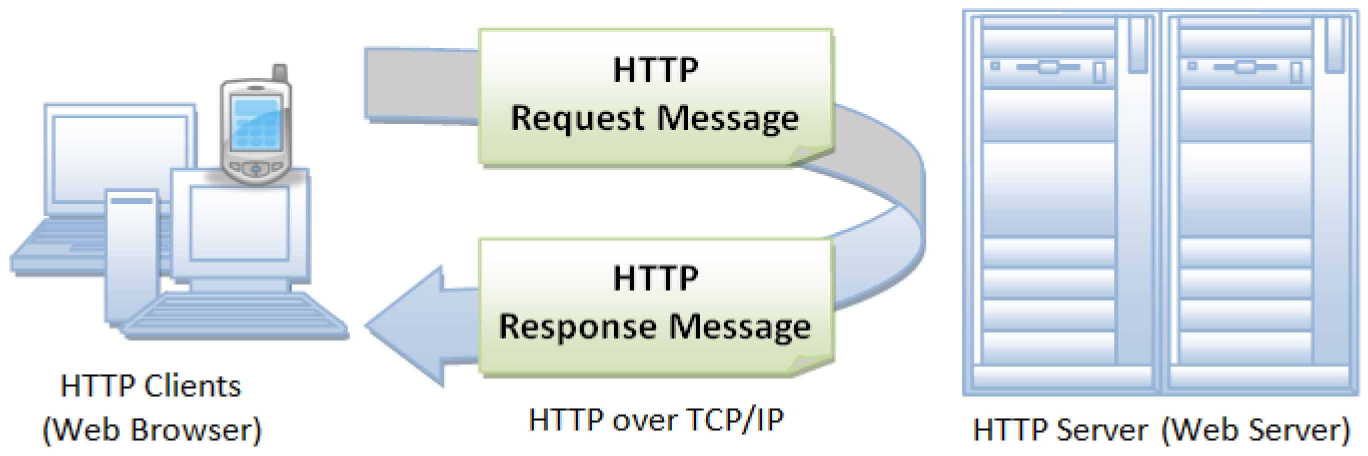


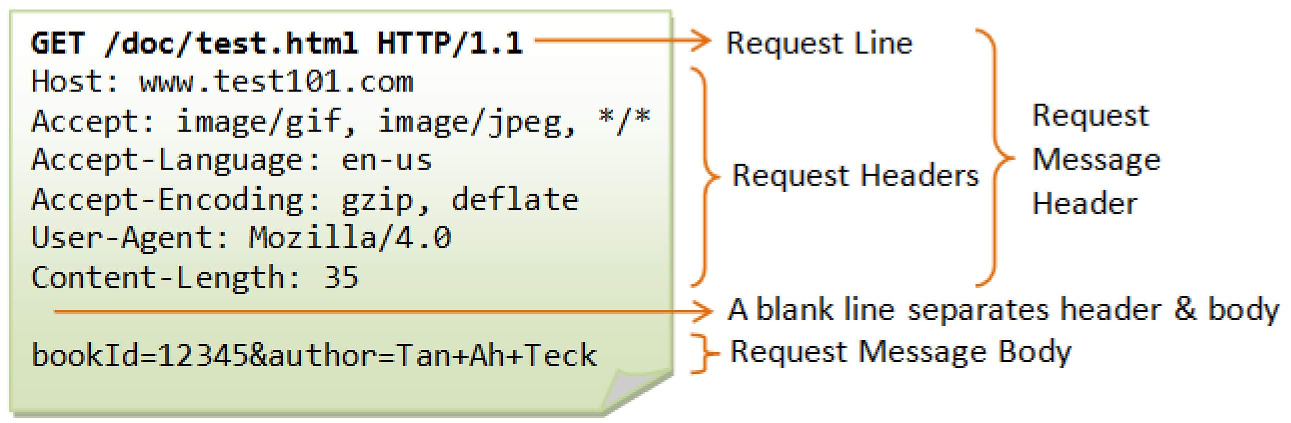
```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

```
hhhhhhhhhhhhh
hhhhhhhhhhhhh
hhhhhhhhhhhhh

bbbbbbbbbbbbb
bbbbbbbbbbbbb
bbbbbbbbbbbbb
bbbbbbbbbbbbb
bbbbbbbbbbbbb
```

HTTP Response Message

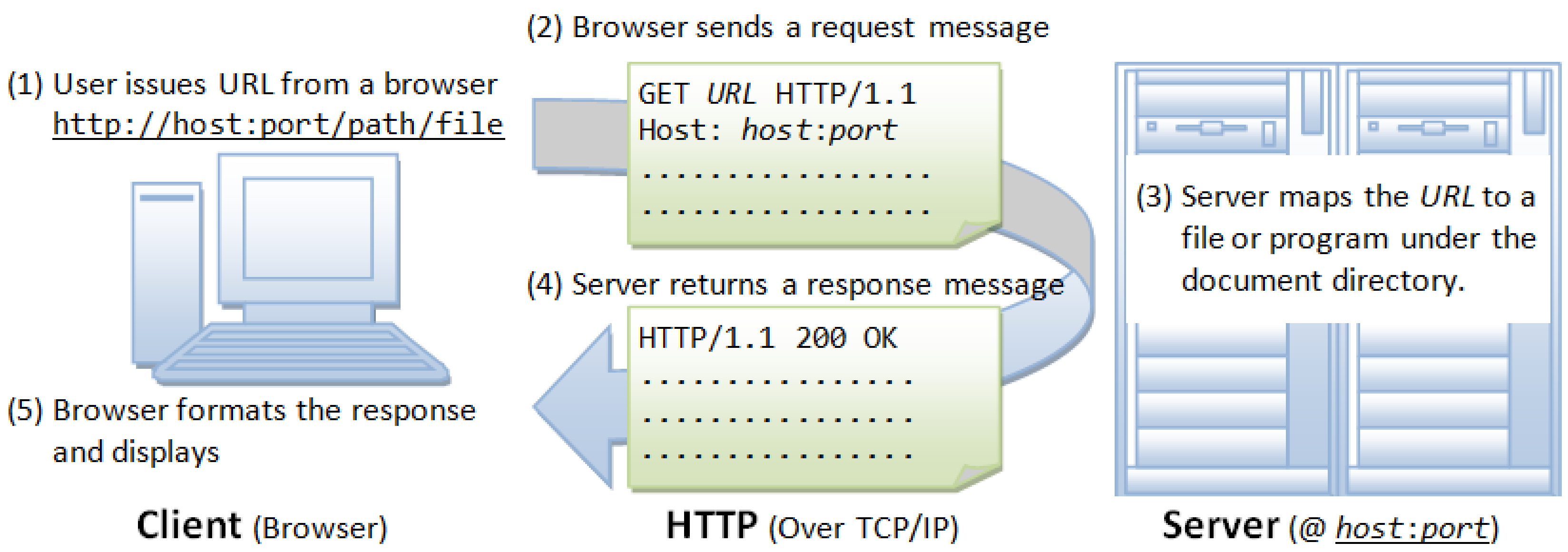
HTTP Request Header





```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

HTTP Request Message



(1) User issues URL from a browser http://host:port/path/file
(2) Browser sends a request message
GET URL HTTP/1.1
Host: host:port
....
....
(3) Server maps the URL to a file or program under the document directory.
(4) Server returns a response message
HTTP/1.1 200 OK
....
....
....
(5) Browser formats the response and displays
Client (Browser)
HTTP (Over TCP/IP)
Server (@ host:port)

HTTP header fields - Response

Examples of response headers are:

- Content-Type: text/html
- Content-Length: 35
- Connection: Keep-Alive
- Keep-Alive: timeout=15, max=100

Examples of status codes are:

- HTTP/1.1 200 OK
- HTTP/1.0 404 Not Found
- HTTP/1.1 403 Forbidden

HTTP header fields - 403

Example: Accessing a Protected Resource

The following GET request tried to access a protected resource. The server returns an error "403 Forbidden". In this example, the directory "htdocs\forbidden" is configured to deny all access in the Apache HTTP server configuration file "httpd.conf" as follows:

```
<Directory "C:/apache/htdocs/forbidden">
Order deny,allow
  deny from all
</Directory>
```

```
GET /forbidden/index.html HTTP/1.0
(blank line)
```

```
HTTP/1.1 403 Forbidden
Date: Sun, 18 Oct 2009 11:58:41 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 222
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access /forbidden/index.html
on this server.</p>
</body></html>
```

HTTP "HEAD" Request Method

HEAD request is similar to GET request. However, the server returns only the response header without the response body, which contains the actual document. HEAD request is useful for checking the headers, such as Last-Modified, Content-Type, Content-Length, before sending a proper GET request to retrieve the document.

The syntax of the HEAD request is as follows: HEAD request-URI HTTP-version

(other optional request headers)

(blank line)

(optional request body)

HEAD /index.html HTTP/1.0

(blank line)

HEAD RESPONSE

HTTP/1.1 200 OK

Date: Sun, 18 Oct 2009 14:09:16 GMT

Server: Apache/2.2.14 (Win32)

Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT

ETag: "10000000565a5-2c-3e94b66c2e680"

Accept-Ranges: bytes

Content-Length: 44

Connection: close

Content-Type: text/html

X-Pad: avoid browser bug

Notice that the response consists of the header only without the body, which contains the actual document.

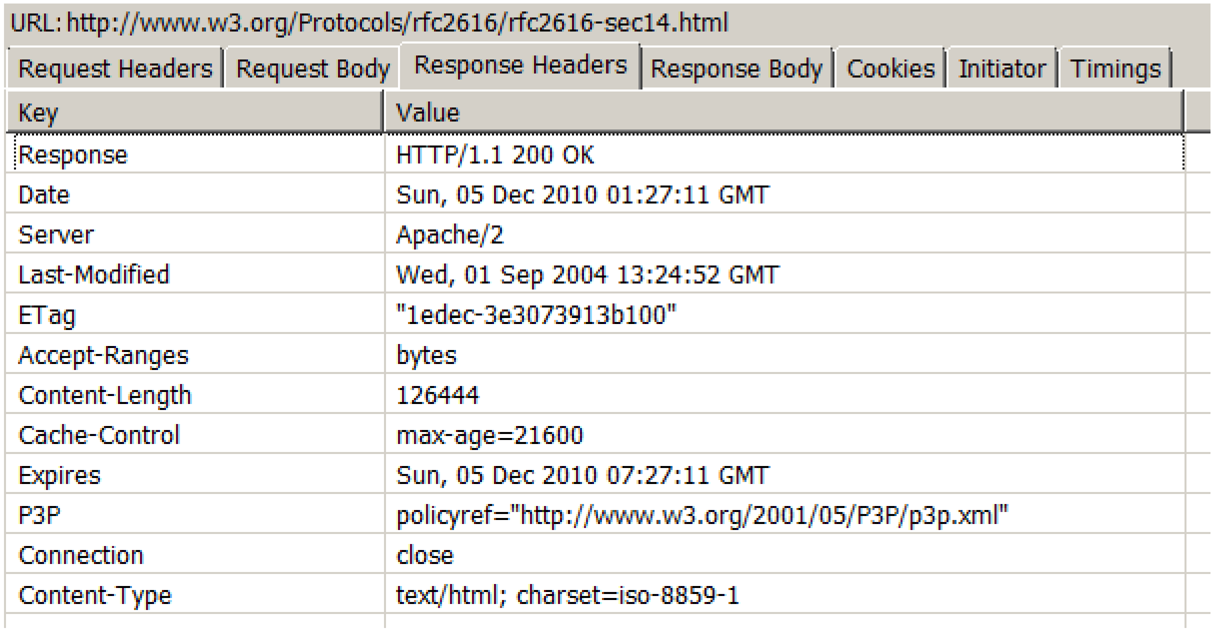
HTTP header fields - Response

Field: Server

Description: A name for the server

Example: Server: Apache/2.4.1 (Unix)

Status: Permanent



URL: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

Request Headers | Request Body | Response Headers | Response Body | Cookies | Initiator | Timings

Key	Value
Response	HTTP/1.1 200 OK
Date	Sun, 05 Dec 2010 01:27:11 GMT
Server	Apache/2
Last-Modified	Wed, 01 Sep 2004 13:24:52 GMT
ETag	"1edec-3e3073913b100"
Accept-Ranges	bytes
Content-Length	126444
Cache-Control	max-age=21600
Expires	Sun, 05 Dec 2010 07:27:11 GMT
P3P	policyref="http://www.w3.org/2001/05/P3P/p3p.xml"
Connection	close
Content-Type	text/html; charset=iso-8859-1

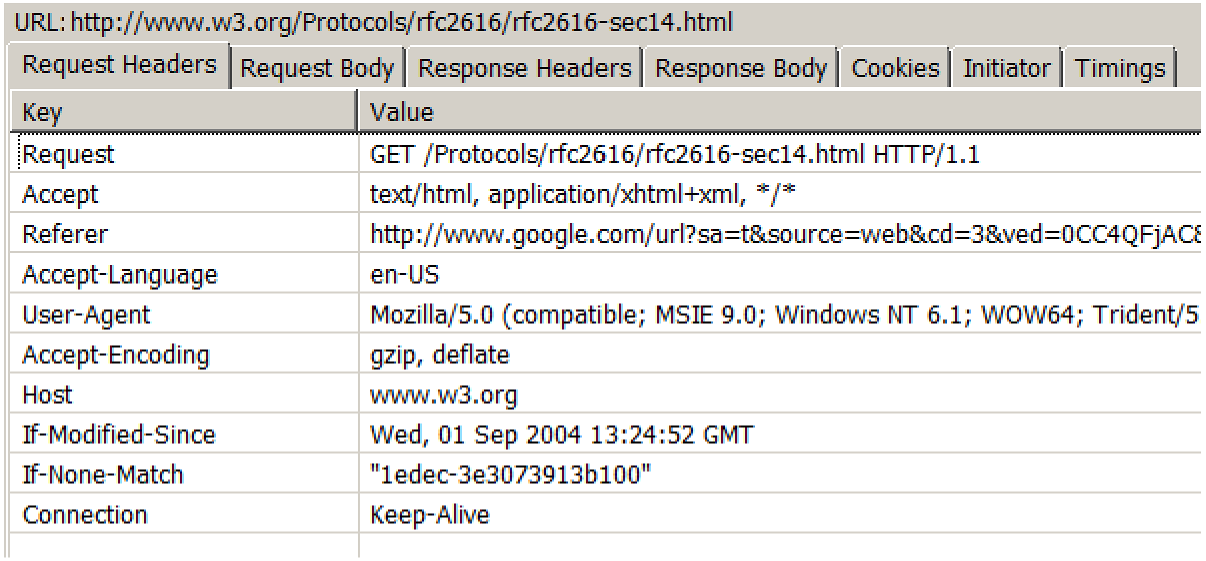
HTTP header fields - Request

Field: User-Agent

Description: The user agent string of the user agent.

Example: Mozilla/5.0 (X11; Linux x86_64; rv:12.0) Gecko/20100101 Firefox/21.0

Status: Permanent



URL: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

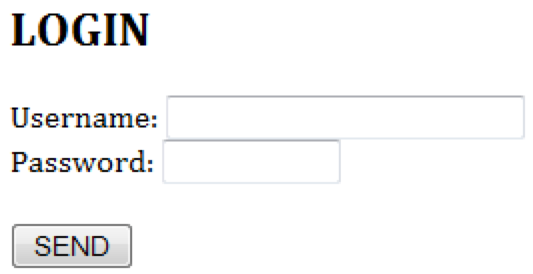
Key	Value
Request	GET /Protocols/rfc2616/rfc2616-sec14.html HTTP/1.1
Accept	text/html, application/xhtml+xml, */*
Referer	http://www.google.com/url?sa=t&source=web&cd=3&ved=0CC4QFjAC
Accept-Language	en-US
User-Agent	Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5
Accept-Encoding	gzip, deflate
Host	www.w3.org
If-Modified-Since	Wed, 01 Sep 2004 13:24:52 GMT
If-None-Match	"1edec-3e3073913b100"
Connection	Keep-Alive

HTTP http GET – form

Example

The following HTML form is used to gather the username and password in a login menu.

```
<html>
<head><title>Login</title></head>
<body>
 <h2>LOGIN</h2>
  <form method="get" action="/bin/login">
    Username: <input type="text" name="user" size="25" /><br />
    Password: <input type="password" name="pw" size="10" /><br /><br />
    <input type="hidden" name="action" value="login" />
    <input type="submit" value="SEND" />
  </form></body>
</html>
```



HTTP http GET – form

The HTTP GET request method is used to send the query string. Suppose the user enters "Peter Lee" as the username, "123456" as password; and clicks the submit button. The following GET request is:

```
GET /bin/login?user=Peter+Lee&pw=123456&action=login HTTP/1.1

Accept: image/gif, image/jpeg, */*

Referer: http://127.0.0.1:8000/login.html

Accept-Language: en-us

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)

Host: 127.0.0.1:8000

Connection: Keep-Alive
```

Note that although the password that you enter does not show on the screen, it is shown clearly in the address box of the browser. You should never use send your password without proper encryption.

http://127.0.0.1:8000/bin/login?user=Peter+Lee&pw=123456&action=login

HTTP http POST – form

Example: Submitting Form Data using POST Request Method

We use the same HTML script as above, but change the request method to POST.

```
<form method="post" action="/bin/login">
  Username: <input type="text" name="user" size="25" /><br />
  Password: <input type="password" name="pw" size="10" /><br /><br />
  <input type="hidden" name="action" value="login" />
  <input type="submit" value="SEND" />
</form>
```

Suppose the user enters "Peter Lee" as username and "123456" as password, and clicks the submit button, the following POST request would be generated by the browser:

```
POST /bin/login HTTP/1.1
Host: 127.0.0.1:8000
Accept: image/gif, image/jpeg, */*
Referer: http://127.0.0.1:8000/login.html
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Content-Length: 37
Connection: Keep-Alive
Cache-Control: no-cache
User=Peter+Lee&pw=123456&action=login
```

HTTP http POST – form

Note that the Content-Type header informs the server the data is URL-encoded (with a special MIME type application/x-www-form-urlencoded), and the Content-Length header tells the server how many bytes to read from the message body.

POST vs GET for Submitting Form Data

As mentioned in the previous section, POST request has the following advantage compared with the GET request in sending the query string:

The amount of data that can be posted is unlimited, as they are kept in the request body, which is often sent to the server in a separate data stream.

The query string is not shown on the address box of the browser.

Note that although the password is not shown on the browser’s address box, it is transmitted to the server in clear text, and subjected to network sniffing. Hence, sending password using a POST request is absolutely not secure.

HTTP Request - Cache Control

The client can send a request header "Cache-control: no-cache" to tell the proxy to get a fresh copy from the original server, even thought there is a local cached copy.

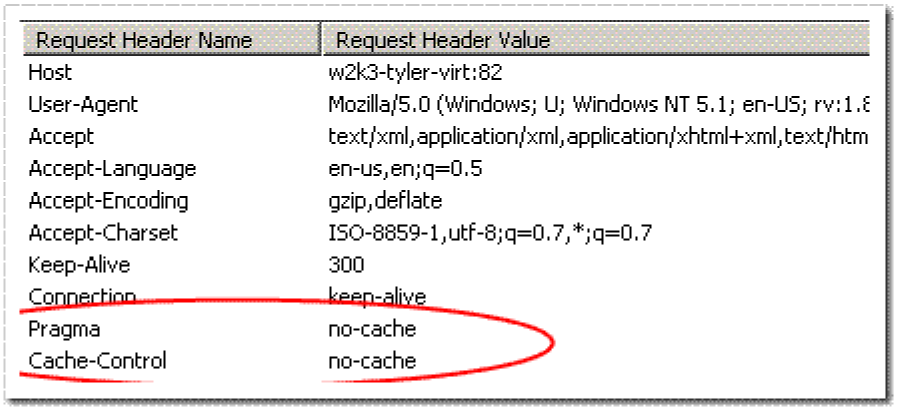
HTTP/1.0 server does not understand this header, but uses an older request header "Pragma: no-cache". You could include both headers in your request.

Pragma: no-cache

Cache-Control: no-cache

HTTP Request - Cache Control

Pragma: no-cache will tell browsers not to cache your content. It doesn't work. In fact there's nothing in the HTTP spec about Pragma being used in Response headers at all. It's supposed to be used in Request headers.



Request Header Name	Request Header Value
Host	w2k3-tyler-virt:82
User-Agent	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8
Accept	text/xml,application/xml,application/xhtml+xml,text/htm
Accept-Language	en-us,en;q=0.5
Accept-Encoding	gzip,deflate
Accept-Charset	ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive	300
Connection	keep-alive
Pragma	no-cache
Cache-Control	no-cache

HTTP – JUHAX.COM

Browser: http://juhax.com

```
GET / HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Language: en-GB
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/51.0.2704.79 Safari/537.36 Edge/14.14393
Accept-Encoding: gzip, deflate
Host: juhax.com
Connection: Keep-Alive
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Last-Modified: Fri, 09 Dec 2016 02:43:21 GMT
Accept-Ranges: bytes
ETag: "12141c1c651d21:0"
Server: Microsoft-IIS/7.5
X-Powered-By: ASP.NET
Date: Wed, 04 Jan 2017 01:48:35 GMT
Content-Length: 7267
```

HTTP – JUHAX.COM

HTTP/200 responses are cacheable by default, unless Expires, Pragma, or Cache-Control headers are present and forbid caching.
HTTP/1.1 ETAG Header is present: "12141c1c651d21:0"
HTTP Last-Modified Header is present: Fri, 09 Dec 2016 02:43:21 GMT

No explicit HTTP Cache Lifetime information was provided.
    Heuristic expiration policies suggest defaulting to: 10% of the delta between Last-Modified and Date.
        That's '2.14:18:31.4000000' so this response will heuristically expire 06/01/2017 17:11:39.

HTTP – JUHAX.COM

```
</body>
<span class="mylinkcode"><div style="display:none">
<a href="http://www.taiwantbl.com/">timberland outlet</a>
<a href="http://www.nb574taiwan.net/">nb574</a>
<a href="http://www.bagintaiwan.com/">gucci bags</a>
<a href="http://www.pumpshoestaiwan.com/">reebok pump</a>
<a href="http://www.kedstaiwan.com/">keds taiwan</a>
<a href="http://www.rb2140taiwan.com/">ray ban 2140</a>
<a href="http://www.beatsstoretaiwan.com/">beats</a>
<a href="http://www.beatsshoptaiwan.com/">beats</a>
<a href="http://www.taiwantblshoes.com/">timberland outlet</a>

<a href="http://www.offthewallshoestaiwan.com/">vans</a>
<a href="http://www.offthewalltaiwan.com/">vans</a>
<a href="http://www.nbtaiwan.net/">nb574</a>
<a href="http://www.pumptaiwan.com/">reebok pump</a>
<a href="http://www.bagshoptaiwan.com/">gucci taiwan</a>
<a href="http://www.prokedsneaker.com/">keds taiwan</a>
<a href="http://www.fashionclothing-mart.com/">cheap moncler jackets</a>
<a href="http://www.fashionclothing-mart.com/">moncler outlet</a>
<a href="http://www.fashionshoes-mart.com/nike-dunks-c-114.html">cheap nike
dunks</a>
</div></span>
</html>
```