HTTP Tutorial

**HTTP tutorial** provides basic and advanced concepts of HTTP (Hyper Text Transfer Protocol). Our HTTP tutorial is developed for beginners and professionals.

What is HTTP

* HTTP stands for Hypertext Transfer Protocol.
* Hypertext Transfer Protocol is a set of rule which is used for transferring the files like, audio, video, graphic image, text and other multimedia files on the WWW (World Wide Web).
* HTTP is an application-level protocol. The communication usually takes place through TCP/IP sockets, but any reliable transport can also be used.
* The standard (default) port for HTTP connection is 80, but other port can also be used.
* The first version of HTTP was HTTP/0.9, which was introduced in 1991.
* The latest version of HTTP is HTTP/3, which was published in September 2019. It is an alternative to its processor HTTP/2.
* This latest version is already in use on the web with the help of UDP (User Datagram Protocol) instead of TCP (Transmission Control Protocol) for the underlying transport protocol.
* HTTP is used to make communication between a variety of hosts and clients. It supports a mixture of network configuration.
* HTTP is a protocol that is used to transfer the hypertext from the client end to the server end, but HTTP does not have any security.
* Whenever a user opens their Web Browser, that means the user indirectly uses HTTP.

Three important things about HTTP

**Connectionless:** HTTP is connectionless. When the HTTP client opens the browser, the browser initiates an HTTP request. After making the request, the client disconnect from the server and wait for the response. When the response is ready, the server re-establish the connection again and delivers the response to the client, after which the client disconnects the connection. So both client and server know about each other during the current request and response only.

**Media Independent:** HTTP is media independent. HTTP can deliver any sort of data, as long as the two computers can read it.

**Stateless:** The HTTP is stateless. The client and server just know about each other just during the current request. If the connection is closed, and two computers want to connect again, they need to provide information to each other anew, and the connection is handled as the very first one.

HTTP Needs

* The HTTP was designed mainly to fetch the html document and send it to the client. That all the HTTP was doing in 1991, and it did not support other media types, it just delivers html document.
* It was designed in an exquisite way, and it was continually evolved, and features were being added to it, it becomes the most convenient way to quickly and reliably move data on the web.

What is HTTPS

* HTTPS stands for Hypertext Transfer Protocol Secure. HTTPS has a secure transfer.
* It was developed by Netscape.
* HTTPS is used to encrypt or decrypt user HTTP page or HTTP page requests that are returned by the webserver.
* HTTPS is first used in HTTP/1.1 and is defined in RFC 2616.
* In HTTPS, the standard port to transfer the information is 443.
* Using the HTTPS, sensitive information that we want to transfer from one user to another user can be done securely.
* HTTPS protocol uses HTTP on connection encrypted by SSL (Secure Socket Layer) or TLS (Transport Layer Security).
* HTTPS protects transmitted data from man-in-the-middle (MITM) attacks and eavesdropping.
* It is the default protocol for conduction financial transactions on the web.

Parameters of HTTP

In this section, we will discuss various HTTP parameters and their syntax. For example, date and time format, character set, etc. These parameters are used in the construction of our request and response message while writing the HTTP program of the client or server.

The various parameters of HTTP are as follows:

HTTP Version

To indicate the version of the protocol, HTTP uses a **<major>.<minor>** numbering scheme. The protocol versioning policy allows the sender to indicate the format of a message and its capacity for understanding further HTTP communication.

The first line in the HTTP-Version field indicates the version of the HTTP message.

**Syntax**

1. HTTP-Version   = "HTTP" "/" 1\*DIGIT "." 1\*DIGIT

**Example**

1. HTTP / 1.1

Entity Tags

Entity tags are used to compare two or more entities from the same requested resource.

**Syntax**

1. entity-tag = [ weak ] opaque-tag
2. weak = "W/"
3. opaque-tag = quoted-string

An Entity tag must be unique across all the entity versions associated with a particular resource.

Date/Time Formats

Date/Time format can be defined in two ways:

**1) Full Date:**

All the date/time stamps of HTTP MUST be represented in Greenwich Mean Time (GMT). HTTP application has three different formats for the representation of date/time stamps:

1. Sun, 06 Nov 1994 08:49:37 GMT ; RFC 822, updated by RFC 1123
2. Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036
3. Sun Nov  6 08:49:37 1994 ; ANSI C's asctime() format

In HTTP, GMT is exactly equal to UTC (Coordinated Universal Time), which was indicated in the first two formats by the inclusion of "GMT".

**2) Delta Second**

Some fields of HTTP header allow a time value, which is specified as an integer number of seconds, which will be in decimal after the time that the message was received.

1. delta-seconds = 1\*DIGIT

Uniform Resource Identifiers (URI)

URI is simply formatted case insensitive string, which contains the name, location, etc. to identify the website or web server.

**Syntax:**

1. http\_URL = "http:" "//" host [ ":" port ] [ abs\_path [ "?" query ]]

Here,

* **"http"** scheme is used to locate network resources through the HTTP protocol.
* If the **port** is empty, the port is assumed to **80**. If **abs\_path** is empty, it is equivalent to anabs\_pathof **"/"**. **Characters** in the "unsafe" and "reserved" sets are equivalent to their **""%" HEX HEX"**encoding.
* Comparisons of host names MUST be case-insensitive.

**Example:**

1. http://abc.com:80/~smith/home.html
2. http://ABC.com/%7Esmith/home.html
3. http://ABC.com:/%7esmith/home.html

Character Set

The Character set is used to specify the character set that the user prefers. Using comma, multiple character sets can be separated. HTTP character sets are identified using the case ?insensitive tokens.

1. charset = token

**Example**

1. ISO-8859-1
3. Or
5. US-ASCII

Content Codings

Content coding values are used to show an encoding transformation that has been applied to an entity. Primarily, content-codings are used to allow a document to e compressed or transformed without losing the information. In content-coding, an entity is stored in coded form, transmitted directly, and only the recipient it encoded.

1. content-coding   = token

All the values of content-coding are case-insensitive. In the Accept-Encoding and Content-Encoding header fields, HTTP/1.1 uses content-coding value.

**Example**

1. Accept-encoding: gzip
3. or
5. Accept-encoding: compress

Transfer Codings

The values of transfer coding are used to indicate an encoding transformation that has been applied to an entity to ensure "safe transport" through the network. Transfer coding is different from content coding. Transfer coding is the property of the message, and it is not the original entity.

**Syntax:**

1. transfer-coding = "chunked" | transfer-extension
2. transfer-extension = token \*( ";" parameter )

In Transfer coding, parameters are in the form of attribute/value pairs.

**Example:**

1. Parameter = attribute "=" value
2. Attribute = token
3. Value = token | quoted-string

All the values of Transfer coding are case-insensitive.

Product Tokens

Product Tokens allow communication applications to identify themselves by the version and name of the software. Product token should be to the point and short.

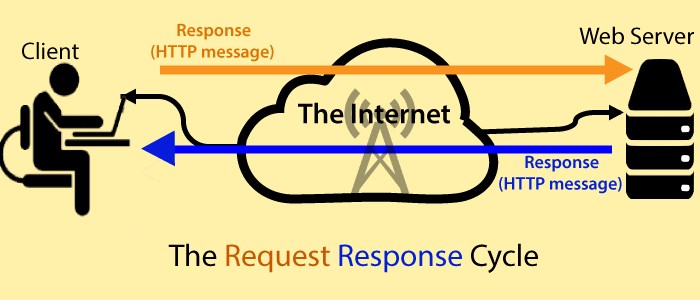
**Syntax**

1. product = token ["/" product-version]
2. product-version = token

**Example**

1. User-Agent: CERN-LineMode/2.15 libwww/2.17b3
2. Server: Apache/0.8.4

Working of WEB

1. We will have a client on the left side and server on the right side. A user wants to see a website, like [www.javatpoint.com/java-tutorial](https://www.javatpoint.com/java-tutorial). The user types the URL of a page using a client program, usually a browser. But first, the computer of the user and the web server need to be physically connected. That is the job of the internet. Using the TCP/IP protocol, it establishes a connection using a combination of cable media or wireless media and does all the necessary work to prepare the environment for the two computers to talk via the HTTP protocol.
2. When the connection establishes, the client sends a request called the HTTP message, but because the HTTP is a connectionless protocol, so the client disconnects from the server and waits for the response.
3. On the other side, the server processes the request, prepare the response, establish the connection again, and send it back the response and again in the form of an HTTP message to the client. Then the two computers completely disconnect.
4. 

HTTP Response

HTTP Response sent by a server to the client. The response is used to provide the client with the resource it requested. It is also used to inform the client that the action requested has been carried out. It can also inform the client that an error occurred in processing its request.

An HTTP response contains the following things:

1. Status Line
2. Response Header Fields or a series of HTTP headers
3. Message Body

In the request message, each HTTP header is followed by a carriage returns line feed (CRLF). After the last of the HTTP headers, an additional CRLF is used and then begins the message body.

Status Line

In the response message, the status line is the first line. The status line contains three items:

**a) HTTP Version Number**

It is used to show the HTTP specification to which the server has tried to make the message comply.

**Example**

1. HTTP-Version = HTTP/1.1

**b) Status Code**

It is a three-digit number that indicates the result of the request. The first digit defines the class of the response. The last two digits do not have any categorization role. There are five values for the first digit, which are as follows:

**Code and Description**

**1xx: Information**

It shows that the request was received and continuing the process.

**2xx: Success**

It shows that the action was received successfully, understood, and accepted.

**3xx: Redirection**

It shows that further action must be taken to complete the request.

**4xx: Client Error**

It shows that the request contains incorrect syntax, or it cannot be fulfilled.

**5xx: Server Error**

It shows that the server failed to fulfil a valid request.

**c) Reason Phrase**

It is also known as the status text. It is a human-readable text that summarizes the meaning of the status code.

An example of the response line is as follows:

1. HTTP/1.1 200 OK

Here,

* HTTP/1.1 is the HTTP version.
* 200 is the status code.
* OK is the reason phrase.

Response Header Fields

The HTTP Headers for the response of the server contain the information that a client can use to find out more about the response, and about the server that sent it. This information is used to assist the client with displaying the response to a user, with storing the response for the use of future, and with making further requests to the server now or in the future.

1. response-header = Accept-Ranges
2. | Age
3. | ETag
4. | Location
5. | Proxy-Authenticate
6. | Retry-After
7. | Server
8. | Vary
9. | WWW-Authenticate

The name of the Response-header field can be extended reliably only in combination with a change in the version of the protocol.

Message Body

The response's message body may be referred to for convenience as a response body.

The body of the message is used for most responses. The exceptions are where a server is using certain status codes and where the server is responding to a client request, which asks for the headers but not the response body.

For a response to a successful request, the body of the message contains either some information about the status of the action which is requested by the client or the resource which is requested by the client. For the response to an unsuccessful request, the body of the message might provide further information about some action the client needs to take to complete the request successfully or about the reason for the error.

HTTP Entity

The Entity is transferred by the HTTP request and response message if not otherwise restricted by the request method or response status code. An entity is used to consist of entity-header fields and entity-body, although some responses will only include the entity-headers.

In this section, both the sender and recipient refer to either the client or the server, which depends on who receives the entity.

Entity Header Fields

Entity-header fields are used to define the metainformation about the entity-body. If body isnot present, the entity-header field defines metainformation about the resource identified by request. Some of this metainformation is OPTIONAL. Some might be REQUIRED by the portion of this specification.

**Syntax**

1. entity-header = Allow
2. | Content-Encoding
3. | Content-Language
4. | Content-Length
5. | Content-Location
6. | Content-MD5
7. | Content-Range
8. | Content-Type
9. | Expires
10. | Last-Modified
11. | extension-header
12. extension-header = message-header

The extension-header mechanism is used to allow the additional entity-header fields, which is defined without the protocol change, but these fields cannot be assumed to be recognizable by the recipient. Recipient ignores the unrecognized header fields and MUST be forwarded by transparent proxies.

Entity Body

If there is any entity-body, it will send with an HTTP request or an HTTP response in the format as described below. The entity-header fields define the encoding.

**Syntax**

1. entity-body    = \*OCTET

When a message-body is present, an entity-body will only present in a message. The entity-body is obtained from the message-body using decoding any Transfer-Encoding that might have been applied to ensure the proper and safe transfer of the message.

Type

When an entity-body is included with a message, the header fields Content-Type and Content-Encoding determines the data type of that body. These are used to define a two-layer, ordered encoding model:

**Syntax**

1. entity-body := Content-Encoding( Content-Type( data ) )

Content-Type is used to specify the media type of the underlying data. Content-Encoding indicates any additional content coding which is applied to the data, usually for the purpose of data compression that is a requested resource property.

Any message of HTTP/1.1 containing an entity-body SHOULD include a Content-Type header field that defines the media type of that body.If and only if a Content-Type field is not given the media type, the recipient MAY attempt to guess the media type using the inspection of its content or the name extension of the URI, which is used to identify the resource. The recipient SHOULD treat it as "application/octet-stream" type if the media type remains unknown.

Entity Length

The message of an entity length is the length of the message body before applying any transfer-coding.

HTTP Methods

For HTTP/1.1, the set of common methods are defined below. This set can be expanded based on the requirements. The name of these methods is case sensitive, and they must be used in uppercase.

Method and Description

**i) GET**

This method retrieves information from the given server using a given URI. GET request can retrieve the data. It can not apply other effects on the data.

**ii) HEAD**

This method is the same as the GET method. It is used to transfer the status line and header section only.

**iii) POST**

The POST request sends the data to the server. For example, file upload, customer information, etc. using the HTML forms.

**iv) PUT**

The PUT method is used to replace all the current representations of the target resource with the uploaded content.

**v) DELETE**

The DELETE method is used to remove all the current representations of the target resource, which is given by URI.

**vi) CONNECT**

This method establishes a tunnel to the server, which is identified by a given URI.

**vii) OPTIONS**

This method describes the options of communication for the target resource.

GET Method

This method is used to retrieve data from a web server using the specifying parameters in the URL portion of the request. This is the main method that is used for document retrieval. The use of the GET method to fetch first.htm is as follows:

1. GET /first.htm HTTP/1.1
2. User-Agent: Mozilla/69.0 (compatible; MSIE5.01; Windows 10)
3. Host: www.javatpoint.com
4. Accept-Language: en-us
5. Accept-Encoding: gzip, deflate
6. Connection: Keep-Alive

The following are the server response against the above GET request:

1. HTTP/1.1 200 OK
2. Date: Wed, 4 Dec 2019 5:15:40 GMT
3. Server: Apache/2.4. 41 (Win32)
4. Last-Modified: Mon, 2 Dec 2019 15:40:30 GMT
5. ETag: "34aa387-d-1568eb00"
6. Vary: Authorization,Accept
7. Accept-Ranges: bytes
8. Content-Length: 55
9. Content-Type: text/html
10. Connection: Closed
11. **<html>**
12. **<body>**
13. **<h1>** First Program**</h1>**
14. **</body>**
15. **</html>**

HEAD Method

This method is the same as the GET method. But in the HEAD method, the server replies with a response line and headers without entity-body. The use of HEAD method to fetch header information about first.htm is as follows:

1. HEAD /first.htm HTTP/1.1
2. User-Agent: Mozilla/69.0 (compatible; MSIE5.01; Windows 10)
3. Host: www.javatpoint.com
4. Accept-Language: en-us
5. Accept-Encoding: gzip, deflate
6. Connection: Keep-Alive

The following are the server response against the above HEAD request:

1. HTTP/1.1 200 OK
2. Date: Wed, 4 Dec 2019 5:15:40 GMT
3. Server: Apache/2.4. 41 (Win32)
4. Last-Modified: Mon, 2 Dec 2019 15:40:30 GMT
5. ETag: "34aa387-d-1568eb00"
6. Vary: Authorization,Accept
7. Accept-Ranges: bytes
8. Content-Length: 55
9. Content-Type: text/html
10. Connection: Closed

Here, we can see that the server does not send any data after the header.

POST Method

This method is used to send some data to the server, for example, update files from data, etc. The use of POST method to send a form data to the server is as follows:

1. POST /cgi-bin/process.cgi HTTP/1.1
2. User-Agent: Mozilla/69.0 (compatible; MSIE5.01; Windows 10)
3. Host: www.jsvatpoint.com
4. Content-Type: text/xml; charset=utf-5
5. Content-Length: 55
6. Accept-Language: en-us
7. Accept-Encoding: gzip, deflate
8. Connection: Keep-Alive
9. <"xml version="1.0" encoding="utf-5">
10. <string xmlns=" https://www.javatpoint.com/">string</string>

The script of server side process.cgi processes the passed data and sends the response which is as follows:

1. HTTP/1.1 200 OK
2. Date: Wed, 4 Dec 2019 5:15:40 GMT
3. Server: Apache/2.4. 41 (Win32)
4. Last-Modified: Mon, 2 Dec 2019 15:40:30 GMT
5. ETag: "34aa387-d-1568eb00"
6. Vary: Authorization,Accept
7. Accept-Ranges: bytes
8. Content-Length: 55
9. Content-Type: text/html
10. Connection: Closed
11. **<html>**
12. **<body>**
13. **<h1>** Request Processed Successfully**</h1>**
14. **</body>**
15. **</html>**

PUT Method

This method requests the server to store the included entity-body at a location that is specified by the given URL. The below example requests the server to save the given entity-body in first.htm at the root of the server.

1. PUT /first.htm HTTP/1.1
2. User-Agent: Mozilla/69.0 (compatible; MSIE5.01; Windows 10)
3. Host: www.javatpoint.com
4. Accept-Language: en-us
5. Connection: Keep-Alive
6. Content-type: text/html
7. Content-Length: 150
8. **<html>**
9. **<body>**
10. **<h1>** First Program**</h1>**
11. **</body>**
12. **</html>**

In first.htm file, the server will store the given entity-body, and it will also send the following response back to the client:

1. HTTP/1.1 201 Created
2. Date: Wed, 4 Dec 2019 5:15:40 GMT
3. Server: Apache/2.4. 41 (Win32)
4. Content-type: text/html
5. Content-length: 30
6. Connection: Closed
7. **<html>**
8. **<body>**
9. **<h1>** The file was created.**</h1>**
10. **</body>**
11. **</html>**

DELETE Method

This method requests the server to delete a file at a location that is specified by the given URL. The below example requests the server to delete the first.htm file at the root of the server:

1. DELETE /first.htm HTTP/1.1
2. User-Agent: Mozilla/69.0 (compatible; MSIE5.01; Windows 10)
3. Host: www.javatpoint.com
4. Accept-Language: en-us
5. Connection: Keep-Alive

After the above example, the server will delete the first.htm file, and it will also send the response back to the client, which is as follows:

1. HTTP/1.1 200 OK
2. Date: Wed, 4 Dec 2019 5:15:40 GMT
3. Server: Apache/2.4. 41 (Win32)
4. Content-type: text/html
5. Content-length: 30
6. Connection: Closed
7. **<html>**
8. **<body>**
9. **<h1>**URL deleted**</h1>**
10. **</body>**
11. **</html>**

CONNECT Method

This method is used by the client. It establishes a network connection to a web server over HTTP. The below example requests a connection with a web server which is running on the host javatpoint.com:

1. CONNECT www.javatpoint.com HTTP/1.1
2. User-Agent: Mozilla/69.0 (compatible; MSIE5.01; Windows 10)

The following example shows that the connection is established with the server, and the response is sent back to the client:

1. HTTP/1.1 200 Connection established
2. Date: Wed, 4 Dec 2019 5:15:40 GMT
3. Server: Apache/2.4. 41 (Win32)

OPTIONS Method

This method is used by the client. It is used to find out the HTTP methods and other options that are supported by a web server. The below example requests a list of methods which is supported by a web server which is running on javatpoint.com:

1. OPTIONS \* HTTP/1.1
2. User-Agent: Mozilla/69.0 (compatible; MSIE5.01; Windows 10)

In the below example, the server will send information which is based on the current configuration of the server:

1. HTTP/1.1 200 OK
2. Date: Wed, 4 Dec 2019 5:15:40 GMT
3. Server: Apache/2.4. 41 (Win32)
4. Allow: GET,HEAD,POST,OPTIONS,TRACE
5. Content-Type: httpd/unix-directory

# Caching in HTTP

HTTP is used for distributed operating systems, where using the response caches, we can improve the performance. The HTTP/1.1 protocol includes a number of elements which is intended to make properly working of caching. Because these elements interact with each other and they are inextricable from other aspects of the protocol, it is useful to describe the HTTP caching design separately from the detailed description of response codes, methods, headers, etc.

The HTTP Caching is basically used to eliminate the need to send the request in many cases. It is also used to eliminate the need to send full responses in many other cases.

In HTTP, cache mechanisms are implicit directives to caches where expiration times and validators are specified by the server. For this purpose, we use the Cache-Control header.

The Cache-Control header is used to allow the client or the server to transmit a variety of directive in either request or response. The default caching algorithm is overridden by these directives. The directives of the caching are specified in a comma-separated list.

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

### **Example**

1. Cache-control: no-cache

The client is using the cache request directives in its HTTP request as follows:

|  |  |
| --- | --- |
| **S.N.** | **Directive of Cache Request and Description** |
| 1 | **no-cache** It indicates that the response which is returned and cannot be used to satisfy a subsequent request to the same URL. |
| 2 | **no-store** It indicates that the client will not store anything about the request of the client or the response of the server. |
| 3 | **max-age = seconds** It is used to indicate that the client is willing to accept a response whose age is not greater than the time which is specified in seconds. |
| 4 | **max-state [=seconds ]** It is used to indicate that the client is willing to accept a response that has exceeded its expiration time. It must not be expired by more than that time, if the seconds are given. |
| 5 | **min-fresh= seconds** It is used to indicate that the client is willing to accept a response whose lifetime of freshness is not less than its current age plus, which is the specified time in seconds. |
| 6 | **no-transform** It does not convert the entity-body. |
| 7 | **only-if-cached** It does not retrieve new data. A document can be send by the cache only if it is in the cache and should not contact the origin-server to see if it exists a newer copy. |

The directives of cache response can be used by the server in its HTTP response as follows:

|  |  |
| --- | --- |
| **S.N.** | **Directive of Cache Response and Description** |
| 1 | **public** It indicates that any cache may be cached the response. |
| 2 | **private** It is used to indicate that some part of the response message or all the response message is intended for a single user. It must not be cached by a shared cache. |
| 3 | **no-cache** It indicates that the response which is returned cannot be used to satisfy a subsequent request to the same URL. |
| 4 | **no-store** It indicates that the client will not store anything about the request of the client or the response of the server. |
| 5 | **no-transform** It does not convert the entity-body. |
| 6 | **must-revalidate** It indicates that the cache must verify the stale document's status before using it, and expired ones should not be used. |
| 7 | **proxy-revalidate** The proxy-revalidate directives and the must-revalidate directive have the same meaning, except that it does not apply to non-shared user agent caches. |
| 8 | **max-age = seconds** It is used to indicate that the client is willing to accept a response whose age is not greater than the time which is specified in seconds. |
| 9 | **s-maxage = seconds** The maximum age specified by the directives overrides the maximum age specified by either the Expires header or the max-age directives. The private cache always ignores the s-manage directives. |

URL Encoding in HTTP

HTTP URLs sent over the internet using the ASCII character-set. In URL-encoding, the following things are performed:

1. Convert all the "unsafe" characters to "%xx", where xx is the ASCII value of the character in hexadecimal.

2. All the spaces changes into pluses.

3. String the name and value together with = and &

**Example**

1. name1=value1&name2=value2&name3=value3

4. For POST submission, the string will be our message body, and for GET submission, the query string will be our message body.

The ASCII symbols of the characters and their replacements are shown in the following table. These replacements of the symbol can be used in the URL before passing it to the server:

|  |  |  |
| --- | --- | --- |
| **Symbol** | **ASCII** | **Replacement** |
| Backspace | 08 | %08 |
| tab | 09 | %09 |
| linefeed | 10 | %0A |
| creturn | 13 | %0D |
| space | 32 | %20 or + |
| ! | 33 | %21 |
| " | 34 | %22 |
| # | 35 | %23 |
| $ | 36 | %24 |
| % | 37 | %25 |
| & | 38 | %26 |
| ' | 39 | %27 |
| ( | 40 | %28 |
| ) | 41 | %29 |
| \* | 42 | \* |
| + | 43 | %2B |
| ' | 44 | %2C |
| - | 45 | - |
| . | 46 | . |
| / | 47 | %2F |
| 0 | 48 | 0 |
| 1 | 49 | 1 |
| 2 | 50 | 2 |
| 3 | 51 | 3 |
| 4 | 52 | 4 |
| 5 | 53 | 5 |
| 6 | 54 | 6 |
| 7 | 55 | 7 |
| 8 | 56 | 8 |
| 9 | 57 | 9 |
| : | 58 | %3A |
| ; | 59 | %3B |
| < | 60 | %3C |
| = | 61 | %3D |
| > | 62 | %3E |
| ? | 63 | %3F |
| @ | 64 | %40 |
| A | 65 | A |
| B | 66 | B |
| C | 67 | C |
| D | 68 | D |
| E | 69 | E |
| F | 70 | F |
| G | 71 | G |
| H | 72 | H |
| I | 73 | I |
| J | 74 | J |
| K | 75 | K |
| L | 76 | L |
| M | 77 | M |
| N | 78 | N |
| O | 79 | O |
| P | 80 | P |
| Q | 81 | Q |
| R | 82 | R |
| S | 83 | S |
| T | 84 | T |
| U | 85 | U |
| V | 86 | V |
| W | 87 | W |
| X | 88 | X |
| Y | 89 | Y |
| Z | 90 | Z |
| [ | 91 | %5B |
| \ | 92 | %5C |
| ] | 93 | %5D |
| ^ | 94 | %5E |
| \_ | 95 | \_ |
| . | 96 | %60 |
| a | 97 | a |
| b | 98 | b |
| c | 99 | c |
| d | 100 | d |
| e | 101 | e |
| f | 102 | f |
| g | 103 | g |
| h | 104 | h |
| i | 105 | i |
| j | 106 | j |
| k | 107 | k |
| l | 108 | l |
| m | 109 | m |
| n | 110 | n |
| o | 111 | 0 |
| p | 112 | p |
| q | 113 | q |
| r | 114 | r |
| s | 115 | s |
| t | 116 | t |
| u | 117 | u |
| v | 118 | v |
| w | 119 | w |
| x | 120 | x |
| y | 121 | y |
| z | 122 | z |
| { | 123 | %7B |
| | | 124 | %7C |
| } | 125 | %7D |
| ~ | 126 | %7E |
|  | 127 | %7F |
|  | > 127 | Encode with "%xx", where "xx" is the ASCII value of the character, in hexadecimal |

# Security of HTTP

HTTP is used to communicate over the internet, so users, information providers, and application developers should be aware of the limitations of security in HTTP/1.1. This section does not provide a definitive solution to the problems mentioned here. It provides some suggestions to reduce security risk.

## Personal information

In HTTP, clients are often privy to a large amount of personal information like: name of the user, email address, passwords, location, Encryption key, etc. We should be careful to prevent unintentional leakage of this personal information of the client via the HTTP protocol to other sources.

### **1. Abuse of Server Log Information**

In this, all the personal data of the user should be stored at the server in an encrypted form.

### **2. Transfer of Sensitive Information**

HTTP cannot regulate the content of data that is transferred. HTTP cannot have any prior method to determine the sensitivity of any particular part of the information within the context of any request.

Revealing any specific software version of the server might allow the server machine to become more vulnerable to attacks against software which contains security holes.

The Proxies which serve as a portal through the firewall of the network should take special precaution about the transfer of header information which is used to identify the hosts behind the firewall.

### **3. Encoding Sensitive Information in URI's**

The source of a link could be private information, so it is strongly recommended that the user be able to select whether or not the field of the referer is sent.

If the page that we refer was transferred with a source protocol, clients should not include a Referer field in an HTTP request.

### **4. Privacy Issues Connected to Accept Headers**

Accept request-headers can reveal the client's information to all servers which are accessed.

## Attacks Based On File and Path Names

The implementation of the origin server of HTTP should be careful to restrict the document, which is returned by HTTP requests to be only that were intended by the server administrators.

**For example,** Microsoft Windows, UNIX, and other operating systems use "--" as a path component which shows a directory level above the current one. On that type of system, an HTTP server MUST disallow any such construct in the Request-URI if it would, otherwise an HTTP server disallow access to a resource those intended to be accessible through the HTTP server.

## DNS Spoofing

HTTP clients rely heavily on the DNS (Domain name service), and are thus generally prone to security attacks, which are based on deliberate mis-association of IP addresses and the name of the DNS. So the client should be careful in assuming the continuing validity of an IP address and DNS name association.

If the clients of HTTP cache the results of hostname lookups to improve the performance, they must observe the TTL information, which was reported by the DNS. When the IP address of the previously accessed server is changed, then the HTTP clients could be spoofed if they do not observe this rule.

## Location Headers and Spoofing

If the multiple organizations are supported by a single server, and any of the organizations do not trust each other, then it must check the Location value and Content-Location headers in the response that are generated under control of said organizations. These organizations are used to make sure that they do attempt to invalidate resources over which they have no authority.

## Authentication Credentials and Idle Clients

User-agent and existing HTTP clients typically retain the information of authentication indefinitely. HTTP/1.1 does not provide any method for a server to direct clients to discard these cached credentials.

To solve this problem, we can encourage the use of idle timeout, password protection, and other methods that reduce the security problems inherent in this problem.

# HTTP Content Negotiation

Most of the responses of HTTP include an entity which contains the information for interpretation by a user. Naturally, it is used to supply the user with the best available entity corresponding to the request. Unfortunately for cache and server, not all users have the same preferences for what is best. That's why HTTP has provisions for several mechanisms for "**content negotiation**". When there are multiple representations available, the process of selecting the best representation for a given response.

Any response which contains an entity-body MAY be subject to negotiation, including error responses.

In HTTP, there are two types of content negotiation, server-driven negotiation, and agent-driven negotiation. Both the negotiations are orthogonal and thus may be used in combination or separately. One method of combination referred to as transparent negotiation which occurs when the origin server provides the information of agent-driven negotiation, which is used by the cache to provide server-driven negotiation for subsequent requests.

## Server-driven Negotiation

When a server-driven negotiation occurs, then the selection of the best representation for a response is made by an algorithm which is located at the server. Based on the available representation of a resource, the selection is based, and the contents. Selection is also based on contents of particular header fields in the request message or on other information, which is pretending to the request (like the network address of the client).

### **Advantages**

1. It is useful when the algorithm for selecting from among the available representations is difficult to describe to the user agent.
2. It is useful when the server desires to send its "best guess" to the client along with the first response.
3. To improve the guess of the server, the user agent may include request header fields that describe its preferences for such a response.

### **Disadvantages**

1. For the server, it is impossible to determine what might be best for any given user accurately. That's why the server would need complete knowledge of both the capabilities of the user agent and the intended use for the response.
2. It complicates an origin server implementation and the algorithms for generating responses to a request.

## Agent-driven Negotiation

When an agent-driven negotiation occurs, the user agent performs the selection of the best representation for a response after receiving an initial response from the origin server. In agent-driven negotiation, the selection is based on a list of available representations of the response, which is included within the header fields or entity-body of the initial response, with each representation identified by its own URI. Selection from a list of representations may be performed manually by the user selection from a generated menu or automatically.

### **Advantages**

1. It is used when the response would vary over commonly-used dimensions when the origin server is unable to determine the capability of a user agent from examining the request.
2. It is used when public caches distribute server load and reduce network usage.

### **Disadvantages**

An agent-driven negotiation suffers when it needs a second request to obtain the best alternate representation.

## Transparent Negotiation

It is a combination of both server-driven negotiation and agent-driven negotiation. If a cache is supplied in the form of a list of available representations of the response and cache completely understood the variance's dimension, then the cache becomes capable of performing server-driven negotiation on behalf of the origin server.

HTTP Status Code

The Server issues an HTTP Status Code in response to a request of the client made to the server. Status code is a 3-digit integer. The first digit of status code is used to specify one of five standard classes of responses. The last two digits of status code do not have any categorization role.

The status codes are divided into 5 parts, as follows:

|  |  |
| --- | --- |
| **S.N.** | **Code and Description** |
| 1 | **1xx: Informational Response** It is used to show that the request was received, and the process is continuing. |
| 2 | **2xx: Successful** It is used to show that the request was successfully received, understood, and accepted. |
| 3 | **3xx: Redirection** It is used to show that further action needs to be taken to complete the request. |
| 4 | **4xx: Client Error** It is used to show that the request contains bad syntax or cannot be fulfilled. |
| 5 | **5xx: Server Error** It is used to show that the server is failed to fulfill an apparently valid request. |

HTTP status codes are extensible. The application of HTTP is not required o understand the meaning of all the registered status code. A list of all the status codes is given below:

1xx: Information

|  |  |
| --- | --- |
| **Message** | **Description** |
| 100 Continue | It is used to show that the client should continue with its request. The interim response informs the client that the request?s initial part has been received. |
| 101 Switching Protocols | It is used to switches the server. |
| 102 Processing | This code is used to show that the server has received and is processing the request. It indicates that no response is available yet. |
| 103 Early Hints | This code is used to return the headers of some responses before the final HTTP message. |

2xx: Successful

|  |  |
| --- | --- |
| **Message** | **Description** |
| 200 OK | This code is used to show that the request is OK. |
| 201 Created | This code shows that the request has been fulfilled, which results in the creation of a new resource. |
| 202 Accepted | This code shows that the request is accepted for processing, but not yet processed completely. |
| 203 Non-authoritative Information | In the entity-header, the information is from a local third party copy. It is not from the original copy. |
| 204 No Content | This code is used to show that the request is processed successfully by the server and not returning any content. |
| 205 Reset Content | This code is used to tell the user agent to reset the document which sent this request. |

3xx: Redirection

|  |  |
| --- | --- |
| **Message** | **Description** |
| 300 Multiple Choices | This code is used to indicate that the multiple options for the resource from which the client may choose. |
| 301 Moved Permanently | This code shows that the URL of the requested resource has been changed permanently. In response, the new URL gives. |
| 302 Found | This code is used to show that the requested page has moved temporarily to a new URL. |
| 303 See Other | This code is used to show that the requested page can be found under another URL using the GET method. |
| 304 No Modified | This code is used for caching purposes. It shows the client that the response has not been modified, so the client can continue to use the same response?s cached version. |
| 305 Use Proxy | This code is used to show that using the proxy; the requested URL must be accessed, which is mentioned in the Location header. |
| 306 Unused | In the previous version, this code is used. This response code is no longer used, and it is just reserved. |
| 307 Temporary Redirect | This code is used to show that the requested page has moved temporarily to a new URL. |

4xx: Client Error

|  |  |
| --- | --- |
| **Message** | **Description** |
| 400 Bad Request | This code is used to indicate that the server did not understand the request due to invalid syntax. |
| 401 Unauthorized | In this code, the requested page needs a username and password. |
| 402 Payment Required | This code reserved for future use. |
| 403 Forbidden | This code is used to show that the access is forbidden to the requested page. |
| 404 No Found | This code is used to show that the server cannot find the requested page. |
| 405 Method Not Allowed | It shows that the request method is not supported for the requested resource. |
| 406 Not Acceptable | It is used to show that the server can only generate a resource that the client does not accept. |
| 407 Proxy Authentication Required | It is used to show that the client must first authenticate itself with the proxy. |
| 408 Request Timeout | This code is used to show that the request took longer than the server was prepared to wait. |

5xx: Server Error

|  |  |
| --- | --- |
| **Message** | **Description** |
| 500 Internal Server Error | This code is used to show that the server has encountered a situation, and it does not know how to handle it. |
| 501 Not Implemented | This code shows that the request was not completed, and the server did not support the functionally required. |
| 502 Bad Gateway | This code shows that the request was not completed, and the server received an invalid response from the upstream server. |
| 503 Service Unavailable | This code shows that the request was not completed, and the server is temporarily overloading or down. |
| 504 Gateway Timeout | It shows that the gateway has timed out. |
| 505 HTTP Version Not Supported | This code is used to show that the server does not support the "http protocol" version. |