

Understanding HTTP Basics

HTTP stands for hypertext transfer protocol and is used to transfer data across the Web.

It is a critical protocol for web developers to understand and because of its widespread use it is also used in transferring data and commands in IOT applications.

The first version of the protocol had only one method, namely GET, which would request a page from a server.

The response from the server was always an HTML page

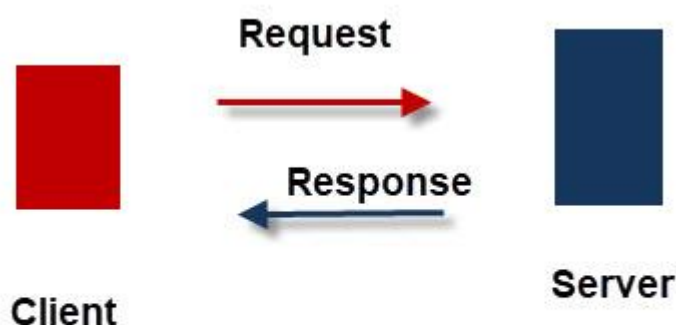
To give you an idea of how simple the HTTP protocol started out take a look at the [Original specification](#) which was only 1 page.

There have been several versions of HTTP starting with the original **0.9 version**.

The current version is 1.1 and was last revised in 2014. See Wiki for more details.

How It Works

Like most of the Internet protocols **http** it is a **command and response text based** protocol using a **client server** communications model.



HTTP Protocol Basics

The client makes a request and the server responds.

The **HTTP protocol** is also a **stateless protocol** meaning that the server isn't required to store session information, and each request is independent of the other.- See this wiki


This means:

- All requests originate at the client (your browser)
- The server responds to a request.
- The requests(commands) and responses are in readable text.
- The requests are independent of each other and the server **doesn't need to track** the requests.

Request and Response Structure

Request and response **message structures** are the same and shown below:

```
generic-message = start-line
                  *(message-header CRLF)
                  CRLF
                  [ message-body ]
```



HTTP Request or Response Message Format

A request consists of:

A command or request + optional headers + optional body content.

A response consists of:

A status code + optional headers + optional body content.

A simple **CRLF** (carriage return and Line feed) combination is used to delimit the parts, and a single blank line (**CRLF**) indicates end of the headers.

If the request or response contains a **message body** then this is **indicated in the header**.

*The presence of a message body in a request is signalled by a **Content-Length** or **Transfer-Encoding** header field. Request message framing is independent of method semantics, even if the method does not define any use for a message body. – RFC 7230 section 3.3.*

Note: the message body is not followed by a CRLF See RFC 7230 section 3.5

HTTP Requests

We saw the general request response format earlier now we will cover the request message in more detail.

The start line is mandatory and is structured as follows:

Method + Resource Path + protocol version

Example if we try to access the web page testpage.htm on www.testsite5.com

The the start line of the request would be

GET /test.htm HTTP/1.1

Where

- **GET** is the method
- **/testpage.htm** is the relative path to the resource.
- **HTTP/1.1** is the protocol version we are using

Notes:

1. A relative path doesn't include the domain name.
2. The web browser uses the URL that we enter to create the **relative URI** of the resource.

Note: URL (uniform resource Locator) is used for web pages. It is an example of a **URI** (uniform resource indicator).

The actual **http request** is not shown by the browser, and is only visible using special tools like **http header live (Firefox)**.

HTTP vs URL

Most people are familiar with entering a url into a web browser. Usually looking like this.



URL Example

The url can also includes the port which is normally hidden by the browser, but you can manually include it as shown below:



URL Example With Port

This tells the web browser the address of the resource to locate and the protocol to use to retrieve that resource (**http**).

http is the transfer protocol that transfer the resource (web page,image,video etc) from the server to the client.

HTTP Responses and Response Codes

Each request has a response. The Response consists of a

- STATUS code And Description
- 1 or more optional headers
- Optional Body message can be many lines including binary data

Response Status codes are split into 5 groups each group has a meaning and a **three digit** code.

- 1xx – Informational
- 2xx – Successful
- 3xx -Multiple Choice
- 4xx– Client Error
- 5xx -Server Error

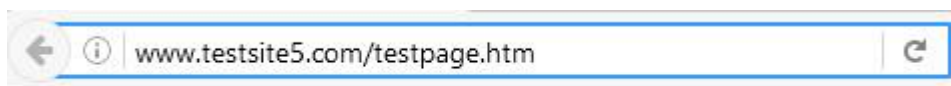
For example a successful page request will return a **200** response code and an unsuccessful a **400** response code.

You can find a complete list and their meaning here

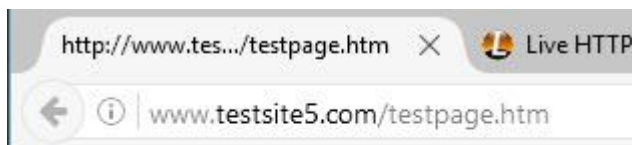
Request Response Example

We are going to examine the requests and response when we access a simple web page (testpage.htm)

Here is what I enter in the browser address bar:



and this is the response that the browser displays:



This is a Test Page

and here is a screen shot of the **http request-response** that happens behind the scenes.

HTTP Headers

http://www.testsite5.com/testpage.htm

GET /testpage.htm HTTP/1.1

Host: www.testsite5.com

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; rv:45.0) Gecko/20100101 Firefox/45.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Connection: keep-alive

HTTP/1.1 200 OK

Date: Wed, 20 Apr 2016 17:14:41 GMT

Server: Apache/2.4.4 (Win32) OpenSSL/0.9.8y PHP/5.4.16

Last-Modified: Wed, 20 Apr 2016 17:13:56 GMT

Etag: "1c-530edb80b9795"

Accept-Ranges: bytes

Content-Length: 28

Keep-Alive: timeout=5, max=100

Connection: Keep-Alive

Content-Type: text/html

Headers

Request

Response

Indicates there is
Body text

Content that is returned is
an html page

HTTP Request-Response

Notice the request headers are automatically inserted by the browser, and so are the response headers are inserted by the web server.

There is no body content in the request. The body content in the reply is a web page, and is shown in the browser, and not by the **live headers tool**.

Request Types

So far we haven't mentioned **request types**, but we have seen the GET request type in our examples.

The **GET request** type or method is used for requesting a resource from a web server.

GET is most commonly used request type and was the only request type in the Original HTTP specification.

Request Types, Methods or Verbs

The HTTP protocol now support 8 request types, also called methods or verbs in the documentation,they are:

- GET – Requesting resource from server
- POST – submitting a resource to a server (e.g. file uploads)
- PUT -As POST but replaces a resource
- DELETE-Delete a resource from a server
- HEAD – As GET but only return headers and not content
- OPTIONS -Get the options for the resource
- PATCH -Apply modifications to a resource
- TRACE -Performs message loop-back

On the Internet today the **GET** (getting web pages) and **POST** (submitting web forms)methods are the ones most commonly used.

HTTP Headers

HTTP headers re used to convey additional information between the client and the server.

Although they are optional they make up the most of the http request and are almost always present.

When you request a web page using a web browser the headers are **inserted automatically** by the web browser, and you don't see them.

Similarly the response headers are inserted by the web server and are not seen by the user.

There are extensions available for both Firefox and Chrome that let you view http headers and also command line tools like curl.

We will look at some example request and response headers later.

Request and Response Header Structure

Request and response headers share a common structure.

They consist of a **header name + colon + header value**. Example

Connection:keep-alive

Name

Value

HTTP-Headers

An header can have multiple values, in which case they are separated using a comma.

Field names are **case insensitive** according to the [RFC 2616](#) but **field values** should be treated as **case sensitive**

There are many headers, and it is not important to be familiar with them all.

There is a really good list of headers with explanation on [tutorialspoint](#)

You should note that only necessary headers are sent all other headers are assumed by the web server and client to be their default.

For example the connection header is not normally sent as the default behaviour is **keep-alive** and this is assumed by the server.

Common Request Headers

Connection Header

The original HTTP protocol used **non persistent** connections.

This meant that the client

1. Made a request
2. Got a Response
3. Closed the Connection

If you consider a TCP/IP connection to be the same as a telephone connection. This means that you:

1. Dial the number and get an answer (connection established).

2. Say something and get an acknowledgement
3. Hang Up.

Because it takes time and resources to **establish the connection** in the first place it makes no sense to drop it so quickly.

Therefore in **HTTP 1.0** the client can tell the server that it will keep the connection open by using the **connection: keep-alive header**.

*In **HTTP v1.1** the default behaviour was changed and **persistent connections** become the **default mode**.*

Now the client can tell the server that it will **close the connection** by using the header **connection: close**.

*This header is not normally sent as the default assumed by the server is **keep-alive**.*

Host Header

Almost all websites, including this one, use shared hosting.

With shared hosting the web server is configured as a virtual host and all virtual hosts will be assigned to a **single IP address**.

The **hosts header** tells the web server which server to refer the request to e.g.

Host:www.steves-internet-guide.com

User-Agent Header

This gives information about the client making the request as shown below:

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:69.0) Gecko/20100101 Fire
```

Accept Request Headers

These headers are used for **content negotiation** and are sent by the client (browser) to the server, and tells the server what formats the client can understand.

The **Accept** header is used to tell the server what **media types** the client prefers e.g. Text, audio etc.

For normal web pages common values are **text/plain** and **text/html**.

e.g.

Accept:text/plain,text/html

For **JSON** encoded data the header

Accept:application/json

is used.

Other accept headers are

- **Accept-Charset:ISO-8859,UTF-8**
- **Accept-Language:en-GB,en**
- **Accept-Encoding:gzip,deflate**

Example Headers from live session:

```
Host: www.steves-internet-guide.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:69.0) (
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
```

Request Headers Example

Common Response Headers

The screen shot below shows the response headers using the curl command requesting a web page from **steves-internet-guide.com**:

```
C:\Users\steve>curl www.steves-internet-guide.com -I
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Keep-Alive: timeout=15
Date: Tue, 17 Sep 2019 14:34:44 GMT
Server: Apache
X-Powered-By: PHP/7.2.22
Vary: Accept-Encoding, Cookie
Cache-Control: max-age=3, must-revalidate
```

Response Headers

The first line of the response is mandatory and consists of the protocol (**HTTP/1.1**), response code (**200**) and description (**OK**).

All subsequent lines are optional

The headers shown are:

CONTENT-Type -This is **Text/html** which is a web page. It also includes the character set which is UTF-8.

Connection – Is keep-alive which means that the connection is held open.

Keep-Alive -This setting as shown is a timeout which says that the server will keep the connection open for **15 seconds** then close it. See [here](#) for more details.

All of the other headers are self explanatory.

Here is another screen shot (partial) of a site returning **JSON data** , notice the Content type.

```
C:\Users\steve>curl jsonplaceholder.typicode.com/posts/1 -I
HTTP/1.1 200 OK
Date: Wed, 18 Sep 2019 16:00:56 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 292
Connection: keep-alive
```

Partial Response Header for Site Returning JSON Data

Setting Headers

For normal users headers are of no interest as they are hidden and created automatically either by the browser (request headers) or web server (response headers).

However web developers and IOT developers will need to be able to set these headers manually.