

Hypertext Transfer Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP) is the protocol that programs use to communicate over the World Wide Web. The main use for the protocol is for the interaction of web browsers and web servers, although there are many other applications!

Communication over ‘The Web’ is facilitated by the use of request and response messages. The browser (client process) sends a request (for information) on behalf of a user and the web server replies with a response (hopefully containing the requested information!). HTTP dictates the style of these request/response messages. The easiest way to get to grips with the protocol is to inspect some specific examples.

Requests.

At the browser (client) we enter a URL to indicate the information we want, e.g.

<http://www.sumnaim.co.uk:80/>

The browser interprets this URL as several fields, as follows:

http://

Use HTTP, the Hypertext Transfer Protocol to generate request messages.

www.sumnaim.co.uk

Connect to the named computer. We could use its IP address instead!

:80

Connect to the remote computer by using port 80. This is the default port number for web transactions so it is normally omitted. We can use any valid port number: 1 through 65535 (although some are in fact reserved for other use as ‘well-known-ports’).

/

This is the document path on the remote computer. This is generally relative to the main document path on the server computer. In this case / would be trying to gain access to this top-level path. The server will interpret this in whatever way it pleases often returning an HTML index document. So we are more likely to use something like /index.html in our URL!

The browser will connect to the remote machine on port 80 by using sockets and a lower level protocol, probably TCP/IP. Client/server programmers therefore need to have some knowledge of socket programming and the use of TCP/IP. On top of the TCP/IP communication we need to set up HTTP messages, so our browser might send a message something like:

```
GET / HTTP/1.1
Accept: image/gif, image/jpeg, text/html
Accept-Language: en-gb
Accept-Encoding: gzip
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT)
Host: www.sumnaim.co.uk
Connection: Keep-Alive
```

Let's consider these 7 lines one at a time:

- Line 1. This is the **Client Request** line. It indicates the function (**method**) that the client expects the server to use to respond to this message – in this case the **GET** method. It also shows what the client wants to retrieve – in this case /, and finally the protocol to be used for the transaction (**HTTP/1.1**). Most servers will allow clients to use the older HTTP/1.0 version of the protocol too!
- Line 2. This is the first of 6 **Header Lines**. It indicates what sort of documents that the client browser can cope with – in this case two image formats and html documents.
- Line 3. Shows that the preferred language is English. This allows a client to specify one or more languages, in the event that the server has the same resource available in multiple languages.
- Line 4. Indicates that the browser can deal with compressed response bodies using gzip.
- Line 5. Identifies the client as Mozilla (version 4.0) running on Windows NT. It also indicates that really the user is accessing MS Internet Explorer v 5.01!).
- Line 6. Tells the server what this browser thinks the server's hostname is. This is mandatory in HTTP/1.1!
- Line 7. Tells the server to keep the underlying TCP connection open until told to disconnect by the client. In HTTP 1.1 the default behaviour to keep the connection open (make it persistent) until the client asks for it to be closed. Under HTTP 1.0 the standard was to close the connection after the client request was completed.
- Line 8. The hidden line that we might forget! Simply a **blank line**.

Responses.

Given a request similar to that shown above, the server looks for a resource associated with '/' and returns it to the client. How the server interprets the resource varies, it may simply return a copy of a static file or it may dynamically generate the information! The resource will be returned in a response message together with associated header fields. In this case we might get a response message similar to:

```
HTTP/1.1 200 OK
Date: Mon, 01 Jan 2006 00:01:20 GMT
Server: Apache/1.3.6 (Unix)
Last-Modified: Sun, 31 Dec 2005 23:49:30 GMT
Etag: "3e7de-123-456bc78a"
Accept-Ranges: bytes
Content-Length: 121
Connection: close
Content-type: text/html
```

```
<title>Greetings!</title>
<h1>Happy New Year!</h1>
Welcome to 2006 may you enjoy good health, happiness and success.
```

The Response Message begins with a series of Header Lines giving information about document and the server itself. A blank line separates these Header Lines and the actual returned document (the body or entity or entity-body!). Let's study this Response Message one line at a time:

- Line 1. Shows the protocol used by the server (HTTP 1.1), followed by a **Status Code** and a **Reason Phrase** (200 OK). HTTP 1.1 defines a range of Status Codes and their corresponding Reason Phrases.
- Line 2. Is the current time on the server using Greenwich Mean Time (GMT).
- Line 3. Indicates the server application that is running i.e. Apache 1.3.6 on Unix.
- Line 4. Specifies the most recent modification time of the resource requested by the client. Often used for caching purposes at the client so the browser need not request the entire resource again unless the modification time changes.
- Line 5. This is an entity tag to provide the client with a unique identifier for the server resource.
- Line 6. Indicates that the server is capable of returning subsections (bytes) of a document, instead of supplying the whole document every time it is requested. This is useful when a client is accessing a database to retrieve records.
- Line 7. Gives the number of bytes (characters) in the entity-body that follows the header lines.
- Line 8. Indicates that the server will close the connection on completion of its response. If the client wants to send another request it must open another connection.
- Line 9. Tells the browser what type of resource the server is supplying in the entity-body – in this case HTML text.
- Line 10. A blank line to separate the Header Lines from the entity-body.

General HTTP Message format.

Client Request.

```
Method URL HTTP-version  
General-header  
Request-header  
Entity-header  
  
Entity-body
```

(Note the **blank line** between the header fields and the Entity-body!)

The HTTP protocol specifies lots of headers - transactions seldom make use of all of them. In fact you could probably use just `GET /index.html HTTP/1.1` without any header lines and still get a sensible response!

Note that HTTP **requests** have the following general components:

1. A method name, which tells the server what function (method) the client expects it to use to satisfy the request. Possible HTTP 1.1 methods are GET, POST, HEAD, PUT, LINK, UNLINK, DELETE, OPTIONS and TRACE.

The URL specifies the location of the resource to apply the method to. Each server will interpret this in its own way but must respond using a standard HTTP 1.1 response message.

The last entry on the first line must specify the HTTP version used by the client.

2. Optional General message Headers indicating such things as the current time.
3. Request Headers to give the server more information about the client, e.g. its identity and the type of resource it can cope with.
4. Entity headers to give information about an entity sent to a server, such as encoding schemes, length data type and origin (most request messages do not have an Entity-body).

Server Response.

```
HTTP-version Status-code Reason-phrase
General-header
Response-header
Entity-header

Entity-body
```

(**Note** the **blank line** between the header fields and the Entity-body!)

In the server response the general headers and entity headers are used in the same way as for a request.

The server will make every effort to conform to the most compatible version of HTTP that the client is using. The status code indicates the result of the request and the reason-phrase is the human readable version of this.

The response headers give the client information about the configuration of the server. It informs the client what methods are supported, requests authorization or tells the client to try again later!

Some general notes on methods and their data.

GET. When a browser sends a GET message it generally has an empty entity-body because it is literally requesting the return of some resource. The name of that resource is supplied as the URL. If the client needs to send some data as well this is placed at the end of the URL by appending a question mark, e.g. <http://hostname/service/getData?page3>, or to see if item 1234 is in stock we might send <http://www.fredscompany.com/stock?item=1234>.

The client often caches the returned resource assuming that the user may want to access it again quite soon!

POST. If the browser has requested a form and the user has completed it then the results may be POSTed back to the server in the entity-body. The browser probably will not cache the resource. It does not expect the user to re-use it in the near future!

Fragments. HTML pages can be divided into subsections or fragments. By using the # symbol a browser can request a fragment. In actual fact it will probably request the whole document and then display from the required fragment onwards, e.g. if we wanted to look at the IDE section of a page displaying disks we might set up the URL <http://joesgroup.com/disks#IDE>. The browser would probably just GET the disks page and then display from IDE onwards.

Query Strings. If a user fills out a form or perhaps needs to interrogate a database at the server end, then the browser sends a Query String. This is appended to the URL by means of a question mark, e.g. suppose we want to query a clothes store to determine if it has any blue Levi jeans with waist 32 inches and long leg length, the URL the browser produces might be:

<http://www.clothes.com/order?item=Levi%20jeans&colour=blue&waist=32&leg=long>

Notice that the query string joins all the fields by means of the ampersand (&). Note also that, since the space character is a reserved symbol, we replace the space with a percentage sign (%) followed by the hex value (20) for the ASCII code for space (the space character has been replaced by an ESCape code).

The client may send the request using GET as its method and an empty body, e.g.

```
GET /order?item=Levi%20jeans&colour=blue&waist=32&leg=long HTTP/1.1
... ..
Host: www.clothes.com
... ..
```

Or, more likely, by using POST as its method with the query string in its body, e.g.

```
POST /order HTTP/1.1
... ..
Host: www.clothes.com
... ..

?item=Levi%20jeans&colour=blue&waist=32&leg=long
```

It is up to the server to cope with this as best it can!

Parameters. The name/value pairs (e.g. waist=32) we have seen to give extra information to a server method are called parameters. The parameters can be supplied to a resource by separating them from the main URL by means of semi-colons, e.g. to indicate that we want to look at e-type jaguars we might use:

<http://www.cars.uk/jags?type=e>

If several parameters are needed, a semi-colon separated list is formed.

Why different techniques?

Different protocol schemes use different URL styles. Not only does HTTP use URLs but so does FTP and SMTP. The URLs vary slightly in their style but there is significant overlap. They all adhere to a general syntax that can be divided into nine (9) general parts:

<scheme>://<user>:<passwd>@<host>:<port>/<path>;<params>?<query>#<frag>

It is up to the client and server processes to make their best attempt at a suitable interpretation of the scheme!

Revision of the general form of messages.

Remember **CR/LF** means Carriage Return followed by Line Feed (in C denoted by \r\n).

Request Message (client to server)

```
Method Space URL Space HTTP-versionCR/LF
General-headerCR/LF
Request-headerCR/LF
Entity-headerCR/LF
CR/LF
Entity-body
```

Response Message (server to client)

```
HTTP-version Space Status-code Space Reason-phraseCR/LF
General-headerCR/LF
Request-headerCR/LF
Entity-headerCR/LF
CR/LF
Entity-body
```

References.

David Gourley and Brian Totty, "*HTTP The Definitive Guide*": O'Reilly 2002.
Clinton Wong, "*HTTP Pocket Reference*": O'Reilly 2000.