HTTP

The **Hypertext Transfer Protocol** (HTTP) is an **application-level protocol** for distributed, collaborative, hypermedia information systems. This is the foundation for data communication for the World Wide Web (i.e. internet) since 1990. HTTP is a generic and stateless protocol which can be used for other purposes as well using extensions of its request methods, error codes, and headers.

HTTP is a **TCP/IP based** communication protocol, that is used to deliver data (HTML files, image files, query results, etc.) on the World Wide Web. **The default port is TCP 80**, but other ports can be used as well. It provides a standardized way for computers to communicate with each other. HTTP specification specifies how clients' request data will be constructed and sent to the server, and how the servers respond to these requests.

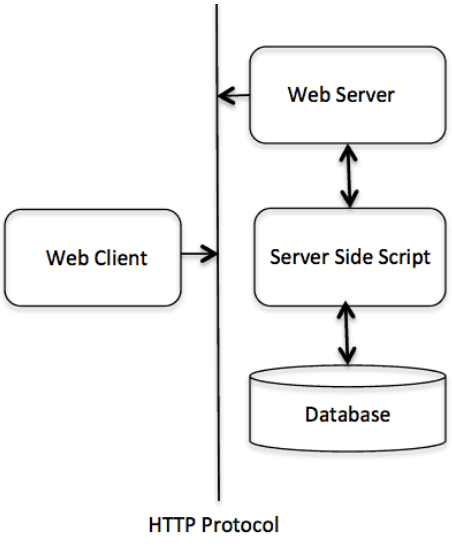
There are three basic features that make HTTP a simple but powerful protocol:

- **HTTP is connectionless:** The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client disconnects from the server and waits for a response. The server processes the request and re-establishes the connection with the client to send a response back.
- **HTTP is media independent:** It means, any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type using appropriate MIME-type.
- **HTTP is stateless:** As mentioned above, HTTP is connectionless and it is a direct result of HTTP being a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.

> HTTP/1.0 uses a new connection for each request/response exchange, where as HTTP/1.1 connection may be used for one or more request/response exchanges.

Basic Architecture

The HTTP protocol is a **request/response protocol** based on the client/server based architecture where web browsers, robots and search engines, etc. act like HTTP clients, and the Web server acts as a server.



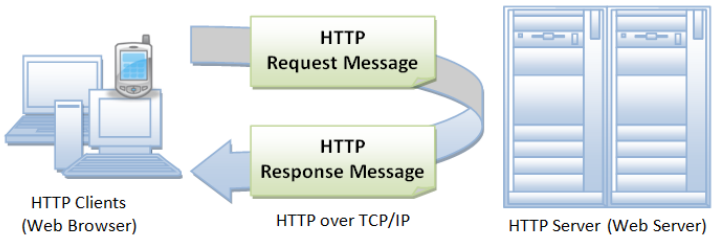
Client

The HTTP client sends a request to the server in the form of a **request method**, **URI**, and **protocol version**, followed by a MIME-like message containing request modifiers, client information, and possible body content over a TCP/IP connection.

Server

The HTTP server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity meta information, and possible entity-body content.

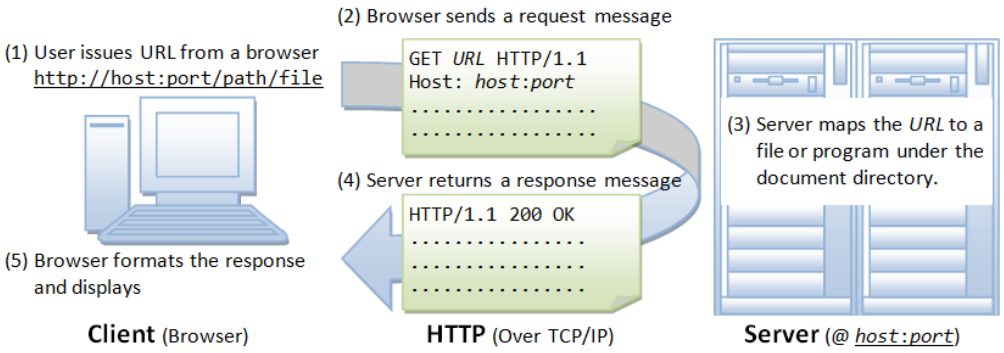
HTTP is an **asymmetric request-response client-server protocol** as illustrated. An HTTP client sends a request message to an HTTP server. The server, in turn, returns a response message. In other words, HTTP is a pull protocol, the client pulls information from the server (instead of server pushes information down to the client).



- HTTP is a stateless protocol. In other words, the current request does not know what has been done in the previous requests.
- HTTP permits negotiating of data type and representation, so as to allow systems to be built independently of the data being transferred.

Browser

Whenever you issue a URL from your browser to get a web resource using HTTP, e.g. http://www.nowhere123.com/index.html, the browser turns the URL into a request message and sends it to the HTTP server. The HTTP server interprets the request message, and returns you an appropriate response message, which is either the resource you requested or an error message. This process is illustrated below:



Uniform Resource Locator (URL)

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:

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

For example, in the URL http://www.nowhere123.com/docs/index.html, the communication protocol is HTTP; the hostname is www.nowhere123.com. The port number was not specified in the URL, and takes on the default number, which is TCP port 80 for HTTP. The path and file name for the resource to be located is "/docs/index.html".

HTTP Protocol

As mentioned, whenever you enter a URL in the address box of the browser, the browser translates the URL into a request message according to the specified protocol; and sends the request message to the server. For example, the browser translated the URL http://www.nowhere123.com/doc/index.html into the following **request message**:

```
GET /docs/index.html HTTP/1.1
Host: www.nowhere123.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
```

When this request message reaches the server, the server can take either one of these actions:

1. The server interprets the request received, maps the request into a file under the server's document directory, and returns the file requested to the client.
2. The server interprets the request received, maps the request into a program kept in the server, executes the program, and returns the output of the program to the client.
3. The request cannot be satisfied, the server returns an error message. 404

An example of the HTTP response message is as shown:

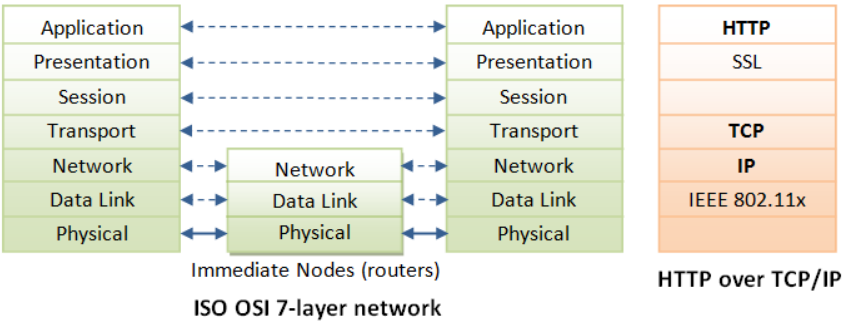
```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

The browser receives the response message, interprets the message and displays the contents of the message on the browser's window according to the media type of the response (as in the Content-Type response header). Common media type include "text/plain", "text/html", "image/gif", "image/jpeg", "audio/mpeg", "video/mpeg", "application/msword", and "application/pdf".

In its idling state, an HTTP server does nothing but listening to the IP address(es) and port(s) specified in the configuration for incoming request. When a request arrives, the server analyzes the message header, applies rules specified in the configuration, and takes the appropriate action. The webmaster's main control over the action of web server is via the configuration, which will be dealt with in greater details in the later sections.

HTTP over TCP/IP

HTTP is a client-server application-level protocol. It typically runs over a TCP/IP connection, as illustrated. (HTTP needs not run on TCP/IP. It only presumes a reliable transport. Any transport protocols that provide such guarantees can be used.)



TCP/IP (Transmission Control Protocol/Internet Protocol) is a set of transport and network-layer protocols for machines to communicate with each other over the network.

IP (Internet Protocol) is a network-layer protocol, deals with network addressing and routing. In an IP network, each machine is assigned an unique IP address (e.g., 165.1.2.3), and the IP software is responsible for routing a message from the source IP to the destination IP. In IPv4 (IP version 4), the IP address consists of 4 bytes, each ranges from 0 to 255, separated by dots, which is called a quad-dotted form. This numbering scheme supports up to 4G addresses on the network. The latest IPv6 (IP version 6) supports more addresses. Since memorizing number is difficult for most of the people, an english-like domain name, such as www.nowhere123.com is used instead. The DNS (Domain Name Service) translates the domain name into the IP address (via distributed lookup tables). A special IP address 127.0.0.1 always refers to your own machine. It's domian name is "localhost" and can be used for local loopback testing.

TCP (Transmission Control Protocol) is a transport-layer protocol, responsible for establish a connection between two machines. TCP consists of 2 protocols: **TCP** and **UDP** (User Datagram Package). TCP is reliable, each packet has a sequence number, and an acknowledgement is expected. A packet will be re-transmitted if it is not received by the receiver. Packet delivery is guaranteed in TCP. UDP does not guarantee packet delivery, and is therefore not reliable. However, UDP has less network overhead and can be used for applications such as video and audio streaming, where reliability is not critical.
TCP multiplexes applications within an IP machine. For each IP machine, TCP supports (multiplexes) up to 65536 ports (or sockets), from port number 0 to 65535. An application, such as HTTP or FTP, runs (or listens) at a particular port number for incoming requests. Port 0 to 1023 are pre-assigned to popular protocols, e.g., HTTP at 80, FTP at 21, Telnet at 23, SMTP at 25, NNTP at 119, and DNS at 53. Port 1024 and above are available to the users.

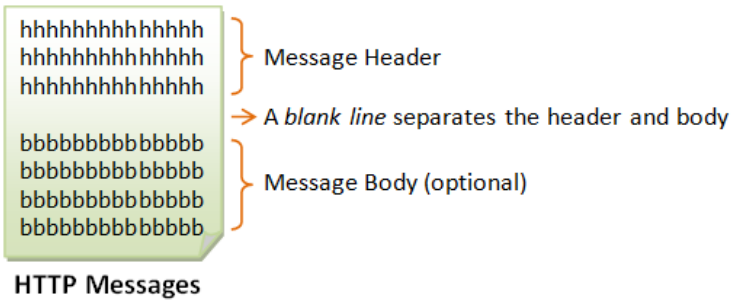
Although TCP port 80 is pre-assigned to HTTP, as the default HTTP port number, this does not prohibit you from running an HTTP server at other user-assigned port number (1024-65535) such as 8000, 8080, especially for test server. You could also run multiple HTTP servers in the same machine on different port numbers. When a client issues a URL without explicitly stating the port number, e.g., http://www.nowhere123.com/docs/index.html, the browser will connect to the default port number 80 of the host www.nowhere123.com. You need to explicitly specify the port number in the URL, e.g. http://www.nowhere123.com:8000/docs/index.html if the server is listening at port 8000 and not the default port 80. In brief, to communicate over TCP/IP, you need to know (a) IP address or hostname, (b) Port number.

HTTP Specifications

The HTTP specification is maintained by **W3C (World-wide Web Consortium)** and available at http://www.w3.org/standards/techs/http.
There are currently **two versions of HTTP**, namely, **HTTP/1.0** and **HTTP/1.1**. The original version, **HTTP/0.9 (1991)**, written by Tim Berners-Lee, is a simple protocol for transferring raw data across the Internet. HTTP/1.0 (1996) (defined in RFC 1945), improved the protocol by allowing MIME-like messages. HTTP/1.0 does not address the issues of proxies, caching, persistent connection, virtual hosts, and range download. These features were provided in HTTP/1.1 (1999) (defined in RFC 2616).

HTTP Request and Response Messages

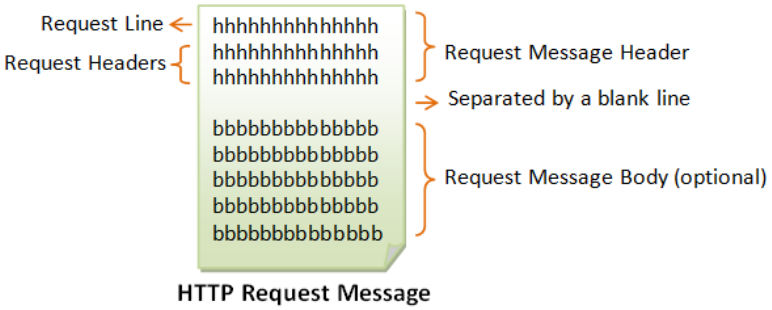
HTTP client and server communicate by sending text messages. The client sends a request message to the server. The server, in turn, returns a response message.
An HTTP message consists of a **message header** and an optional **message body**, separated by a blank line, as illustrated below:



HTTP Messages

HTTP Request Message

The format of an HTTP request message is as follow:



HTTP Request Message

Request Line

The first line of the header is called the request line, followed by optional request headers. The request line has the following syntax:

```
request-method-name request-URI HTTP-version
```

- **request-method-name**: HTTP protocol defines a set of request methods, e.g., GET, POST, HEAD, and OPTIONS. The client can use one of these methods to send a request to the server.
- **Request-URI [*unified resource identifier*]**: specifies the resource requested.
- **HTTP-version**: Two versions are currently in use: HTTP/1.0 and HTTP/1.1.

Examples of request line are:

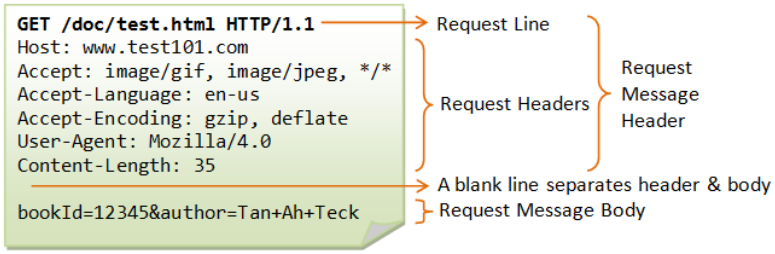
```
GET /test.html HTTP/1.1
HEAD /query.html HTTP/1.0
POST /index.html HTTP/1.1
```

Request Headers

The request headers are in the form of name:value pairs. Multiple values, separated by commas, can be specified.

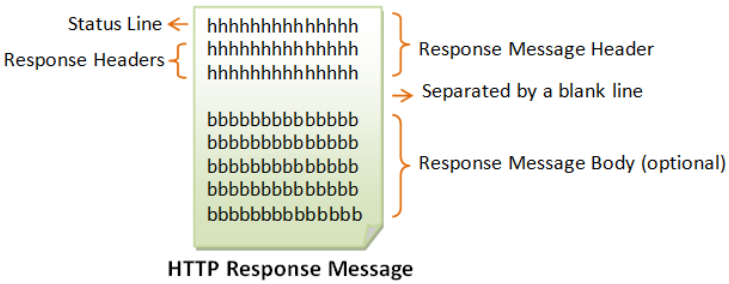
```
request-header-name: request-header-value1, request-header-value2, ...
```

The following shows a sample HTTP request message:



HTTP Response Message

The format of the HTTP response message is as follows:



Status Line

The first line is called the status line, followed by optional response header(s). The status line has the following syntax:

```
HTTP-version status-code reason-phrase
```

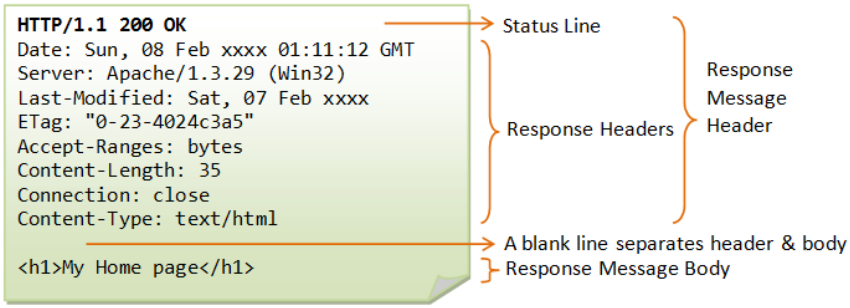
- *HTTP-version*: The HTTP version used in this session. Either HTTP/1.0 and HTTP/1.1.
- *status-code*: a 3-digit number generated by the server to reflect the outcome of the request.
- *reason-phrase*: gives a short explanation to the status code.
- Common status code and reason phrase are "200 OK", "404 Not Found", "403 Forbidden", "500 Internal Server Error".

Response Headers

The response headers are in the form name:value pairs:

```
response-header-name: response-header-value1, response-header-value2, ...
```

The following shows a sample response message:



HTTP Request Methods

HTTP protocol defines a set of request methods. A client can use one of these request methods to send a request message to an HTTP server.
The methods are:

- **GET**: A client can use the GET request to get a web resource from the server.
- **HEAD**: A client can use the HEAD request to get the header that a GET request would have obtained. Since the header contains the last-modified date of the data, this can be used to check against the local cache copy.
- **POST**: Used to post data up to the web server. The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server
- **PUT**: Ask the server to store the data. The PUT method replaces all current representations of the target resource with the request payload.
- **DELETE**: Ask the server to delete the data.
- **TRACE**: Ask the server to return a diagnostic trace of the actions it takes.
- **OPTIONS**: Ask the server to return the list of request methods it supports.
- **CONNECT**: Used to tell a proxy to make a connection to another host and simply reply the content, without attempting to parse or cache it. This is often used to make SSL connection through the proxy.
- Other extension methods.

"GET" Request Method

GET is the most common HTTP request method. A client can use the GET request method to request (or "get") for a piece of resource from an HTTP server. A GET request message takes the following syntax:

```
GET request-URI HTTP-version
(optional request headers)
(blank line)
(optional request body)
```

- The keyword GET is case sensitive and must be in uppercase.
- **request-URI**: specifies the path of resource requested, which must begin from the root "/" of the document base directory.
- **HTTP-version**: Either HTTP/1.0 or HTTP/1.1. This client negotiates the protocol to be used for the current session. For example, the client may request to use HTTP/1.1. If the server does not support HTTP/1.1, it may inform the client in the response to use HTTP/1.0.
- The client uses the optional request headers (such as Accept, Accept-Language, and etc) to negotiate with the server and ask the server to deliver the preferred contents (e.g., in the language that the client preferred).
- GET request message has an optional request body which contains the query string (to be explained later).

HTTP (HyperText Transfer Protocol) is a protocol that utilizes **TCP** to transfer its information between computers (usually Web servers and clients). The client makes an **HTTP** request to the Web server using a Web browser, and the Web server sends the requested information (website) to the client.
Remember, **IP** is required to connect all networks; **TCP** is a mechanism that allows us to transfer data safely; and **HTTP**, which utilizes **TCP** to transfer its data, is a specific protocol used by Web servers and clients.

HTTP/1.0 GET Request

The following shows the response of an HTTP/1.0 GET request (issue via telnet or your own network program - assuming that you have started your HTTP server):

```
GET /index.html HTTP/1.0
(enter twice to create a blank line)
```

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 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

<html><body><h1>It works!</h1></body></html>

Connection to host lost.
```

In this example, the client issues a GET request to ask for a document named "/index.html"; and negotiates to use HTTP/1.0 protocol. A blank line is needed after the request header. This request message does not contain a body.
The server receives the request message, interprets and maps the request-URI to a document under its document directory. If the requested document is available, the server returns the document with a response status code "200 OK". The response headers provide the necessary description of the document returned, such as the last-modified date (Last-Modified), the MIME type (Content-Type), and the length of the document (Content-Length). The response body contains the requested document. The browser will format and display the document according to its media type (e.g., Plain-text, HTML, JPEG, GIF, and etc.) and other information obtained from the response headers.

Notes:

- The request method name "GET" is case sensitive, and must be in uppercase.
- If the request method name was incorrectly spelt, the server would return an error message "501 Method Not Implemented".
- If the request method name is not allowed, the server will return an error message "405 Method Not Allowed". E.g., DELETE is a valid method name, but may not be allowed (or implemented) by the server.
- If the request-URI does not exist, the server will return an error message "404 Not Found". You have to issue a proper request-URI, beginning from the document root "/". Otherwise, the server would return an error message "400 Bad Request".
- If the HTTP-version is missing or incorrect, the server will return an error message "400 Bad Request".
- In HTTP/1.0, by default, the server closes the TCP connection after the response is delivered. If you use telnet to connect to the server, the message "Connection to host lost" appears immediately after the response body is received. You could use an optional request header "Connection: Keep-Alive" to request for a persistent (or keep-alive) connection, so that another request can be sent through the same TCP connection to achieve better network efficiency. On the other hand, HTTP/1.1 uses keep-alive connection as default.

Response Status Code

The first line of the response message (i.e., the status line) contains the response **status code**, which is generated by the server to indicate the outcome of the request.
The status code is a 3-digit number:

- **1xx** (Informational): Request received, server is continuing the process.
- **2xx** (Success): The request was successfully received, understood, accepted and serviced.
- **3xx** (Redirection): Further action must be taken in order to complete the request.
- **4xx** (Client Error): The request contains bad syntax or cannot be understood.
- **5xx** (Server Error): The server failed to fulfill an apparently valid request.

Example: Keep-Alive Connection

By default, for HTTP/1.0 GET request, the server closes the TCP connection once the response is delivered. You could request for the TCP connection to be maintained, (so as to send another request using the same TCP connection, to improve on the network efficiency), via an optional request header "**Connection: Keep-Alive**". The server includes a "Connection: Keep-Alive" response header to inform the client that he can send another request using this connection, before the keep-alive timeout. Another response header "Keep-Alive: timeout=x, max=x" tells the client the timeout (in seconds) and the maximum number of requests that can be sent via this persistent connection.

Encoded URL

URL cannot contain special characters, such as blank or '~'. Special characters are encoded, in the form of %xx, where xx is the ASCII hex code. For example, '~' is encoded as %7e; '+' is encoded as %2b. A blank can be encoded as **%20** or '**+**'. The URL after encoding is called encoded URL.

GET – Safe, Idempotent, Cacheable
PUT – Idempotent
DELETE – Idempotent
HEAD – Safe, Idempotent
POST

Idempotent means the request can be done multiple times
Safe there are no side effects performing that action

Post has non of these characteristics