HTTP - Overview

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.

Basically, 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.

Basic Features

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.

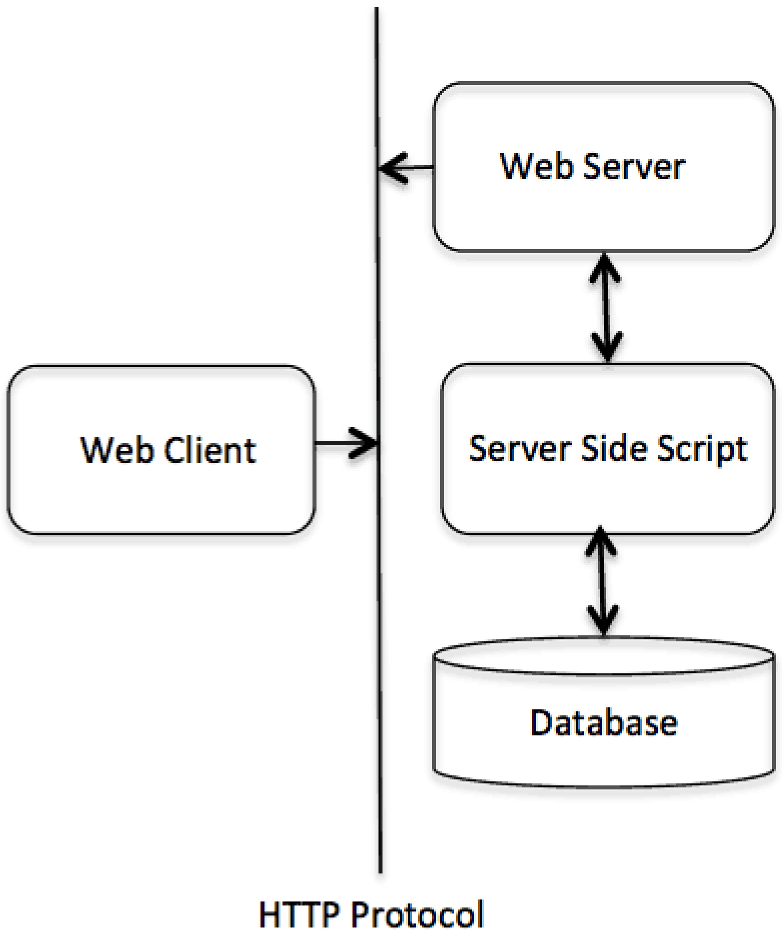
- **HTTP is media independent:** It means, any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type using appropriate MIME-type.

- **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 following diagram shows a very basic architecture of a web application and depicts where HTTP sits:



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.

Uniform Resource Identifiers

Uniform Resource Identifiers (URI) are simply formatted, case-insensitive string containing name, location, etc. to identify a resource, for example, a website, a web service, etc. A general syntax of URI used for HTTP is as follows:

```
URI = "http:" "//" host [ ":" port ] [ abs_path [ "?" query ]]
```

Here if the **port** is empty or not given, port 80 is assumed for HTTP and an empty**abs_path** is equivalent to an **abs_path** of "/". The characters other than those in the**reserved** and **unsafe** sets are equivalent to their ""%" HEX HEX" encoding.

Example

The following three URIs are equivalent:

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

HTTP - Messages

HTTP is based on the client-server architecture model and a stateless request/response protocol that operates by exchanging messages across a reliable TCP/IP connection.

An HTTP "client" is a program (Web browser or any other client) that establishes a connection to a server for the purpose of sending one or more HTTP request messages. An HTTP "server" is a program (generally a web server like Apache Web Server or Internet Information Services IIS, etc.) that accepts connections in order to serve HTTP requests by sending HTTP response messages.

HTTP makes use of the Uniform Resource Identifier (URI) to identify a given resource and to establish a connection. Once the connection is established, **HTTP messages** are passed in a format similar to that used by the Internet mail [RFC5322] and the Multipurpose Internet Mail Extensions (MIME) [RFC2045]. These messages include **requests** from client to server and **responses** from server to client which will have the following format:

```
HTTP-message   = <Request> | <Response> ; HTTP/1.1 messages
```

HTTP requests and HTTP responses use a generic message format of RFC 822 for transferring the required data. This generic message format consists of the following four items.

Message Start-Line

A start-line will have the following generic syntax:

```
start-line = Request-Line | Status-Line
```

We will discuss Request-Line and Status-Line while discussing HTTP Request and HTTP Response messages respectively. For now, let's see the examples of start line in case of request and response:

```
GET /hello.htm HTTP/1.1     (This is Request-Line sent by the client)


HTTP/1.1 200 OK             (This is Status-Line sent by the server)
```

Header Fields

HTTP header fields provide required information about the request or response, or about the object sent in the message body. There are four types of HTTP message headers:

- **General-header:** These header fields have general applicability for both request and response messages.

- **Request-header:** These header fields have applicability only for request messages.

- **Response-header:** These header fields have applicability only for response messages.

- **Entity-header:** These header fields define meta information about the entity-body or, if no body is present, about the resource identified by the request.

All the above mentioned headers follow the same generic format and each of the header field consists of a name followed by a colon (**:**) and the field value as follows:

```
message-header = field-name ":" [ field-value ]
```

Following are the examples of various header fields:

```
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
Host: www.example.com
Accept-Language: en, mi
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain
```

Message Body

The message body part is optional for an HTTP message but if it is available, then it is used to carry the entity-body associated with the request or response. If entity body is associated, then usually **Content-Type** and **Content-Length** headers lines specify the nature of the body associated.

A message body is the one which carries the actual HTTP request data (including form data and uploaded, etc.) and HTTP response data from the server (including files, images, etc.). Shown below is the simple content of a message body:

```
<html>
```

```
   <body>


      <h1>Hello, World!</h1>


   </body>
</html>
```

HTTP Request

An HTTP client sends an HTTP request to a server in the form of a request message which includes following format:

```
•  A Request-line
```

```
•  Zero or more header (General|Request|Entity) fields followed by CRLF
```

```
•  An empty line (i.e., a line with nothing preceding the CRLF)

•  indicating the end of the header fields
```

```
•  Optionally a message-body
```

The following sections explain each of the entities used in an HTTP request message.

Request-Line

The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. The elements are separated by space SP characters.

```
Request-Line = Method SP Request-URI SP HTTP-Version CRLF
```

Let's discuss each of the parts mentioned in the Request-Line.

Request Method

The request **method** indicates the method to be performed on the resource identified by the given **Request-URI**. The method is case-sensitive and should always be mentioned in uppercase. The following table lists all the supported methods in HTTP/1.1.

S.N.	Method and Description
1	**GET** The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
2	**HEAD** Same as GET, but it transfers the status line and the header section only.
3	**POST** A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
4	**PUT** Replaces all the current representations of the target resource with the uploaded content.
5	**DELETE** Removes all the current representations of the target resource given by URI.
6	**CONNECT**

	Establishes a tunnel to the server identified by a given URI.
7	**OPTIONS** Describe the communication options for the target resource.
8	**TRACE** Performs a message loop back test along with the path to the target resource.

Request-URI

The Request-URI is a Uniform Resource Identifier and identifies the resource upon which to apply the request. Following are the most commonly used forms to specify an URI:

```
Request-URI = "*" | absoluteURI | abs_path | authority
```

S.N.	Method and Description
1	The asterisk ***** is used when an HTTP request does not apply to a particular resource, but to the server itself, and is only allowed when the method used does not necessarily apply to a resource. For example: **OPTIONS * HTTP/1.1**
2	The **absoluteURI** is used when an HTTP request is being made to a proxy. The proxy is requested to forward the request or service from a valid cache, and return the response. For example: **GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1**
3	The most common form of Request-URI is that used to identify a resource on an origin server or gateway. For example, a client wishing to retrieve a resource directly from the origin server would create a TCP connection to port 80 of the host "www.w3.org" and send the following lines: **GET /pub/WWW/TheProject.html HTTP/1.1** **Host: www.w3.org**

	Note that the absolute path cannot be empty; if none is present in the original URI, it MUST be given as "/" (the server root).

Request Header Fields

We will study General-header and Entity-header in a separate chapter when we will learn HTTP header fields. For now, let's check what Request header fields are.

The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers.Here is a list of some important Request-header fields that can be used based on the requirement:

- Accept-Charset
- Accept-Encoding
- Accept-Language
- Authorization
- Expect
- From
- Host
- If-Match
- If-Modified-Since
- If-None-Match
- If-Range
- If-Unmodified-Since
- Max-Forwards

- Proxy-Authorization
- Range
- Referer
- TE
- User-Agent

You can introduce your custom fields in case you are going to write your own custom Client and Web Server.

HTTP Response

The following sections explain each of the entities used in an HTTP response message.

Message Status-Line

A Status-Line consists of the protocol version followed by a numeric status code and its associated textual phrase. The elements are separated by space SP characters.

```
Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

HTTP Version

A server supporting HTTP version 1.1 will return the following version information:

```
HTTP-Version = HTTP/1.1
```

Status Code

The Status-Code element is a 3-digit integer where first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit:

S.N.	**Code and Description**

1	**1xx: Informational** It means the request was received and the process is continuing.
2	**2xx: Success** It means the action was successfully received, understood, and accepted.
3	**3xx: Redirection** It means further action must be taken in order to complete the request.
4	**4xx: Client Error** It means the request contains incorrect syntax or cannot be fulfilled.
5	**5xx: Server Error** It means the server failed to fulfill an apparently valid request.

HTTP status codes are extensible and HTTP applications are not required to understand the meaning of all registered status codes. A list of all the status codes has been given in a separate chapter for your reference.

Response Header Fields

We will study General-header and Entity-header in a separate chapter when we will learn HTTP header fields. For now, let's check what Response header fields are.

The response-header fields allow the server to pass additional information about the response which cannot be placed in the Status- Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.

- Accept-Ranges
- Age
- ETag
- Location

- Proxy-Authenticate
- Retry-After
- Server
- Vary
- WWW-Authenticate

You can introduce your custom fields in case you are going to write your own custom Web Client and Server.