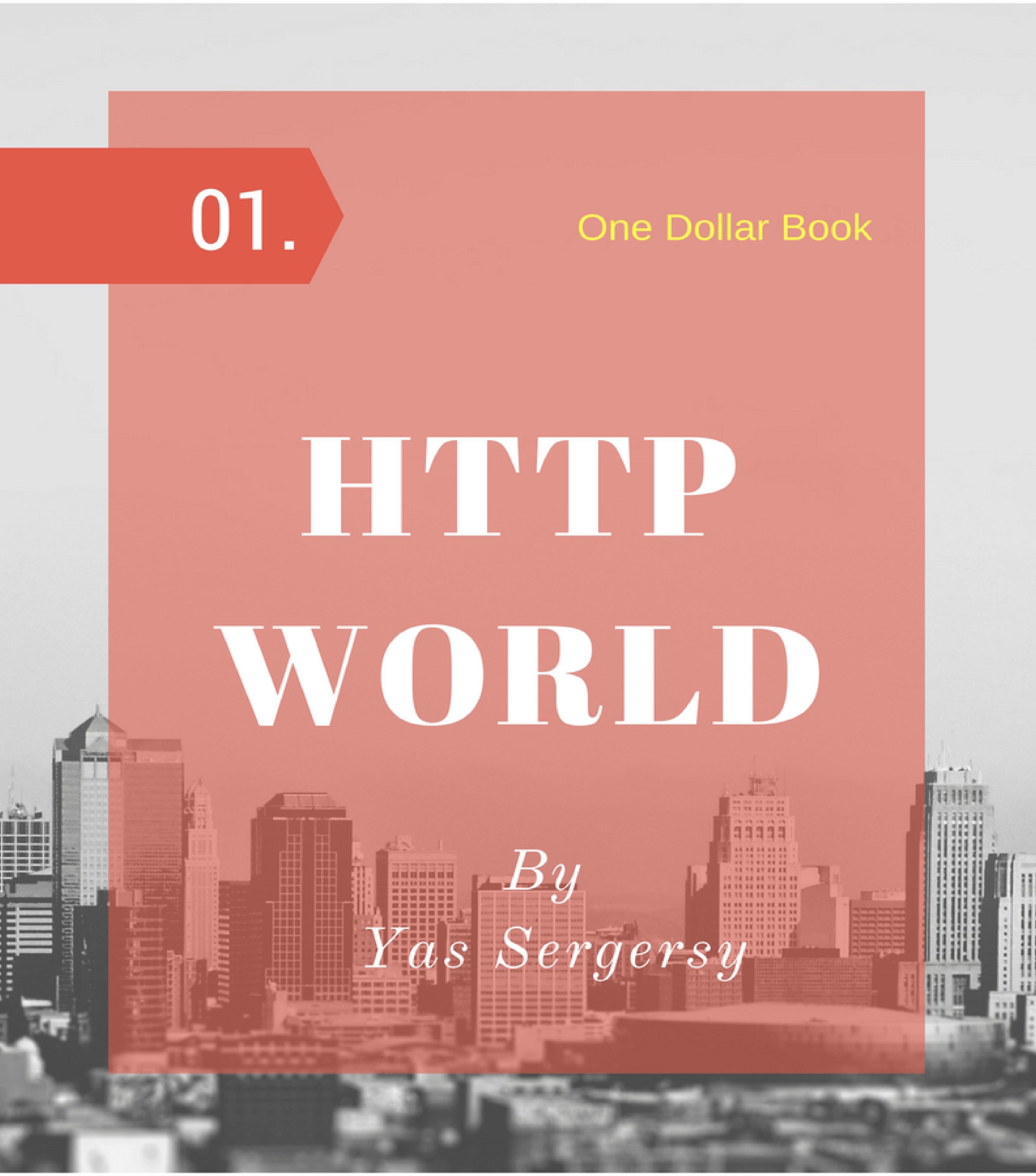
01.

One Dollar Book

HTTP WORLD

By

Yas Sergersy



HTTP World

Yasser Gersy

This book is for sale at http://leanpub.com/HTTPWorld

This version was published on 2016-09-16

Leanpub

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2016 Yasser Gersy

All you need to know about HTTP and HTTPS.By purchasing you are free to copy any content in the book.

Contents

Foreword

This book covers all you need to know about Hyper-Text-Transfer-Protocol.

The best way to learn

is simply by Reading and practise , just try to read and implement what you learned even if you failed , make sure you have just made progress.

What is it

HTTP is just a protocol was specefied to determine the way , the systems talk to each other and transfer data between them by the end of reading this book you will have an inclusive idea of the whole infrastructure of WEB . If you are not familiar with HTTP , do not worry , i kept that in mind , there is a part for begginers talking about HTTP from scratch and another one advanced you can read it after ending the first part .

Dedication

This book and it's contents were collected and arranged for better and fast understanding , the value of science is Priceless , the price you paid is not the price of the contents , it is the price of hours of continuous work and night watching in ordering and collecting information to be in your hand in it‘s simplest way.

The author.

Intro

Thank you for purchasing this book, I hope you have as much fun reading it as I did researching and writing it.

OneDollar HTTP is my second book, meant to help you get started on developing web apps , web servers and pentesting by understanding the infrastructure of the whole WEB. I began writing this as a self-published explanation of HTTP, a by-product of my own learning and collecting. It quickly turned into so much more.

My hope for the book, at the very least, is to open your eyes to the vast world of WEB. At best, I hope this will be your first step towards developing new products.

How It All Started

in 2012 , i was developing a desktop application , and i needed to make online check if the open port is visible or not, of course i searched google and ended at stackoverflow it was so easy , i found the solution copy and paste the application works correctly amazing!, but i was not confortable what i should do , i must read the code and understand what it does , i started looking ,it makes an HTTP request , what is it? back to google and started searching again about web and HTTP , then i turned into learning web development using PHP and first i should learn about HTML i know , what else? i should know more about HTTP and how to request a resoure and how to respond to HTTP Requests , HTTP is used in many applications in our lives all web applications use it and others like desktop ,ATMS, mobile apps and so many , since that i found that i must learn more about HTTP , i tried to collect each word every idea , all the information i find all documents i read , i decided to share with you on this book . So to build a basic knowledge about HTTP , all you have to do is to get a cup of tea and start reading , any example you meet make an implementation to that will help you to study well.

Who This Book Is Written For

This book is written with new developers in mind. It doesn't matter if you're a web developer, web designer,junior hacker , stay at home mom, a 10 year old or a 75 year old. I want this book to be an authoritative reference for understanding the infrastructure of the web , how to send request, how to respond , how to build your own server.

That said, I didn't write this book to preach to the masses. This is really a book about learning much about web and HTTP. As such, I share knowledge.

Thanks

I want to say thanks to my partner peter yaworsk , i learned much from him , while writing our book togetther and some of this books contents are contributed by peter.

Word of Warning

Yes , this book has collected contents from many sources , i tried to collect every thing you need to know , but does not mean the end , you may need to look out for something , just do it , never stop. Good luck!!

Chapter Overview

Chapter 1
Introduction to the web and HTTP.

Chapter 2
History of HTTP .

Chapter 3
sessions.

Chapter 4
Authentication like digest and web forms.

Chapter 5
Methods Like GET , POST and OPTIONS.

Chapter 6
Status Codes.

Chapter 7
keep Alive.

Chapter 8
session state.

Chapter 9
Secured connections.

Chapter 10 Request.

Chapter 11 Response .

Chapter 12 Example shows the implementations of HTTP.

Chapter 13 Similar Protocols like HTTP that appearend recently .

Chapter 14 Tools that help you to build web servers and web applications.

Chapter 15 Resources and refernces that we used in the book.

Chapter 16 Glossary.

Background

If you're starting out fresh like I was and this book is among your first steps into the world of Web Applications or web hacking, it's going to be important for you to understand how the internet works. Before you turn the page, what I mean is how the URL you type in the address bar is mapped to a domain, which is resolved to an IP address, etc.

What is Internet?

To frame it in a sentence: the internet is a bunch of systems that are connected and sending messages to each other. Some only accept certain types of messages, some only allow messages from a limited set of other systems, but every system on the internet receives an address so that people can send messages to it. It's then up to each system to determine what to do with the message and how it wants to respond.

To define the structure of these messages, people have documented how some of these systems should communicate in Requests for Comments (RFC). As an example, take a look at HTTP. HTTP defines the protocol of how your internet browser communicates with a web server. Because your internet browser and web server agreed to implement the same protocol, they are able to communicate.

Scenario

When you enter http://www.google.com in your browser's address bar and press return, the following steps describe what happens on a high level:

- Your browser extracts the domain name from the URL, www.google.com.
- Your computer sends a DNS request to your computer's configured DNS servers. DNS can help resolve a domain name to an IP address, in this case it resolves to 216.58.201.228. Tip: you can use dig A www.google.com from your terminal to look up IP addresses for a domain.
- Your computer tries to set up a TCP connection with the IP address on port 80, which is used for HTTP traffic. Tip: you can set up a TCP connection by running nc 216.58.201.228 80 from your terminal.
- It it succeeds, your browser will send an HTTP request like:

```
GET / HTTP/1.1
Host: www.google.com
Connection: keep-alive
Accept: application/html, */*
```

Request from browser

- Now it will wait for a response from the server, which will look something like:

```
HTTP/1.1 200 OK
Content-Type: text/html

<html>
  <head>
    <title>Google.com</title>
  </head>
  <body>
    .other data .
  </body>
</html>
```

- Your browser will parse and render the returned HTML, CSS, and JavaScript. In this case, the home page of Google.com will be shown on your screen.

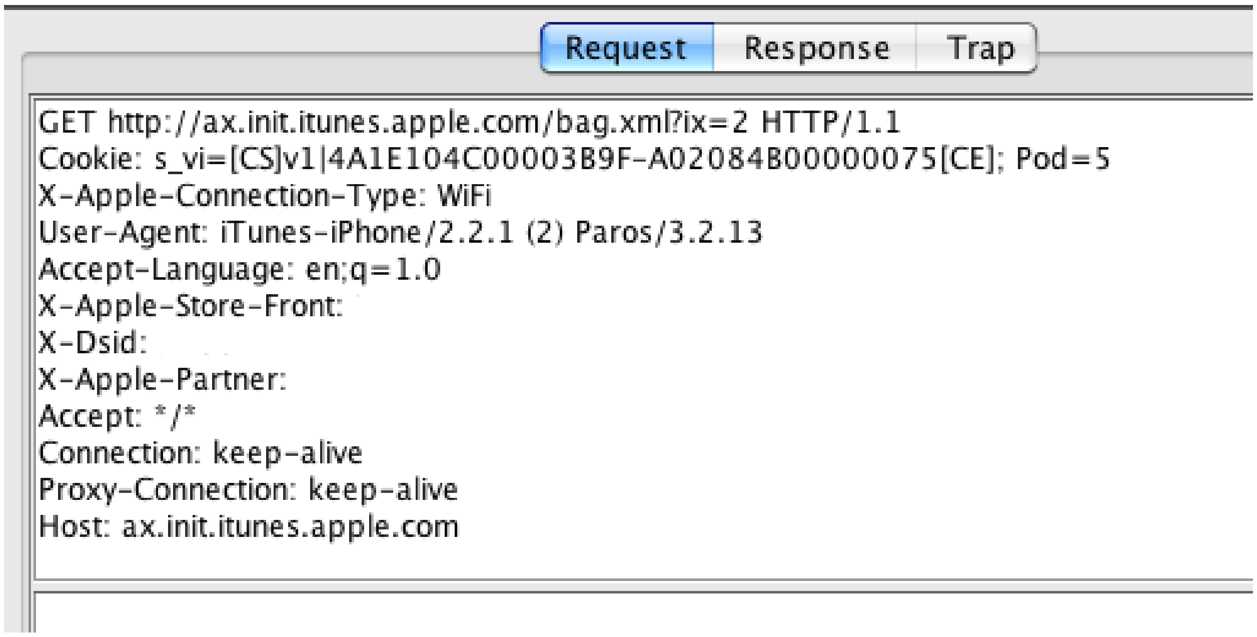
Day 4 of the 2016 Doodle Fruit Games! Find out more at g.co/fruit
Google
Google Search
I'm Feeling Lucky

Browser rendered the HTML,css and javascript codes returned in response above

Now, when dealing specifically with the browser, the internet and HTML, as mentioned previously, there is an agreement on how these messages will be sent, including the specific methods used and the requirement for a Host request-header for all HTTP/1.1 requests, as noted above in bullet 4. The methods defined include GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT and OPTIONS.

The GET method

Means to retrieve whatever information is identified by the request Uniform Request Identifier (URI). The term URI may be confusing, especially given the reference to a URL above, but essentially, for the purposes of this book, just know that a URL is like a person's address and is a type of URI which is like a person's name (thanks Wikipedia). While there are no HTTP police, typically GET requests should not be associated with any data altering functions, they should just retrieve and provide data.

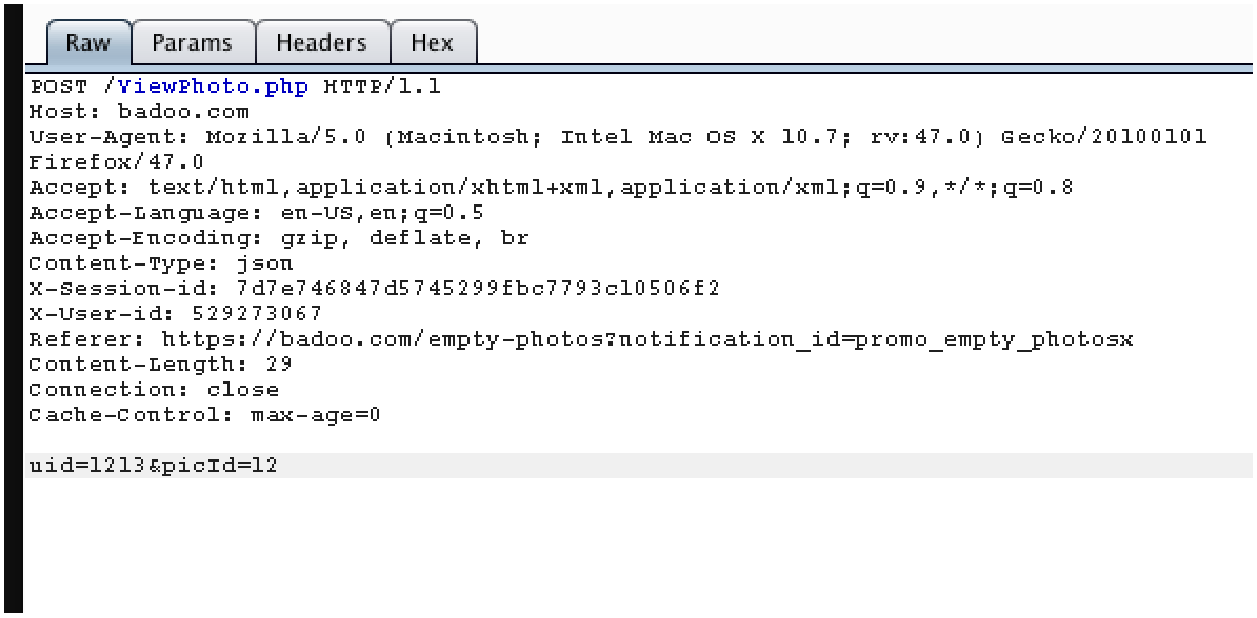


```
GET http://ax.init.itunes.apple.com/bag.xml?ix=2 HTTP/1.1
Cookie: s_vi=[CS]v1|4A1E104C00003B9F-A02084B00000075[CE]; Pod=5
X-Apple-Connection-Type: WiFi
User-Agent: iTunes-iPhone/2.2.1 (2) Paros/3.2.13
Accept-Language: en;q=1.0
X-Apple-Store-Front:
X-Dsid:
X-Apple-Partner:
Accept: */*
Connection: keep-alive
Proxy-Connection: keep-alive
Host: ax.init.itunes.apple.com
```

Get Request

The POST method

Is used to invoke some function to be performed by the server, as determined by the server. In other words, typically there will be some type of back end action performed like creating a comment, registering a user, deleting an account, etc. The action performed by the server in response to the POST can vary and doesn't have to result in action being taken. For example, if an error occurs processing the request.



```
POST /ViewPhoto.php HTTP/1.1
Host: badoo.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:47.0) Gecko/20100101
Firefox/47.0
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, br
Content-Type: json
X-Session-id: 7d7e746847d5745299fbc7793c10506f2
X-User-id: 529273067
Referer: https://badoo.com/empty-photos?notification_id=promo_empty_photosx
Content-Length: 29
Connection: close
Cache-Control: max-age=0

uid=1213&picId=12
```

Post request used while loggin in facebook

Now, armed with a basic understanding of how the internet works, we can dive into the HTTP and Web Application and different aspects.

introduction to HTTP

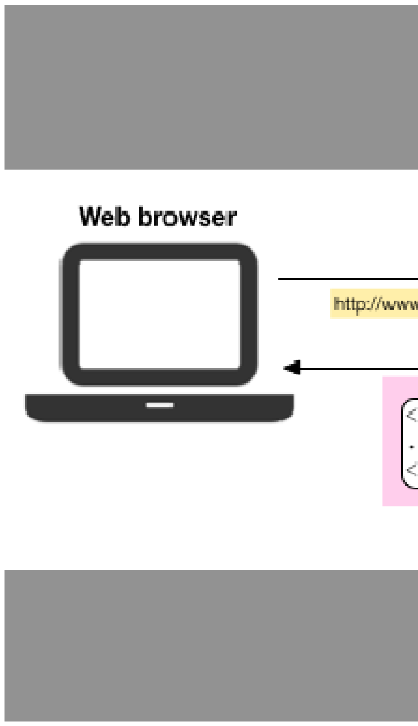
The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.

Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext.

Development of HTTP was initiated by Tim Berners-Lee at CERN in 1989. Standards development of HTTP was coordinated by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C), culminating in the publication of a series of Requests for Comments (RFCs). The first definition of HTTP/1.1, the version of HTTP in common use, occurred in RFC 2068 in 1997, although this was obsoleted by RFC 2616 in 1999.

A later version, the successor HTTP/2, was standardized in 2015, then supported by major web browsers (firefox ,opera) and already supported by major web servers (apache , IIS).

URL beginning with the HTTP scheme and the WWW domain name label.



This image shows What is HTTP , simply sedning a request and receive a response .

Technical overview

URL beginning with the HTTP scheme and the WWW domain name label.HTTP functions as a request - response protocol in the client â€“ server computing model.

For example, may be the client and an application running on a computer hosting a web site may be the server. The client submits an HTTP request message to the server. The server, which provides resources such as HTML files and other content(Text file , music , movies ,etc), or performs other functions on behalf of the client, returns a response message to the client. The response contains completion status information about the request and may also contain requested content in its message body.

A web browser Is an example of a user agent . Other types of user agent include the software used by search providers (web crawlers), voice browsers, mobile apps, and other software that accesses, consumes, or displays web content.

HTTP is designed to permit intermediate network elements to improve or enable communications between clients and servers. High-traffic websites often benefit from web cache servers that deliver content on behalf of upstream servers to improve response time. Web browsers cache previously accessed web resources and reuse them when possible to reduce network traffic. HTTP proxy servers at private network boundaries can facilitate communication for clients without a globally routable address, by relaying messages with external servers.

HTTP is an application layer protocol designed within the framework of the Internet Protocol Suite. Its definition presumes an underlying and reliable transport layer protocol,[2] and Transmission

Control Protocol (TCP) is commonly used. However HTTP can be adapted to use unreliable protocols such as the User Datagram Protocol (UDP), for example in HTTPU and Simple Service Discovery Protocol (SSDP).

HTTP resources are identified and located on the network by uniform resource locators (URLs), using the uniform resource identifier (URI) schemes http and https. URIs and hyperlinks in Hypertext Markup Language (HTML) documents form inter-linked hypertext documents.

HTTP/1.1 is a revision of the original HTTP (HTTP/1.0). In HTTP/1.0 a separate connection to the same server is made for every resource request. HTTP/1.1 can reuse a connection multiple times to download images, scripts, stylesheets, etc after the page has been delivered. HTTP/1.1 communications therefore experience less latency as the establishment of TCP connections presents considerable overhead.

Common web browsers

Basic Features

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

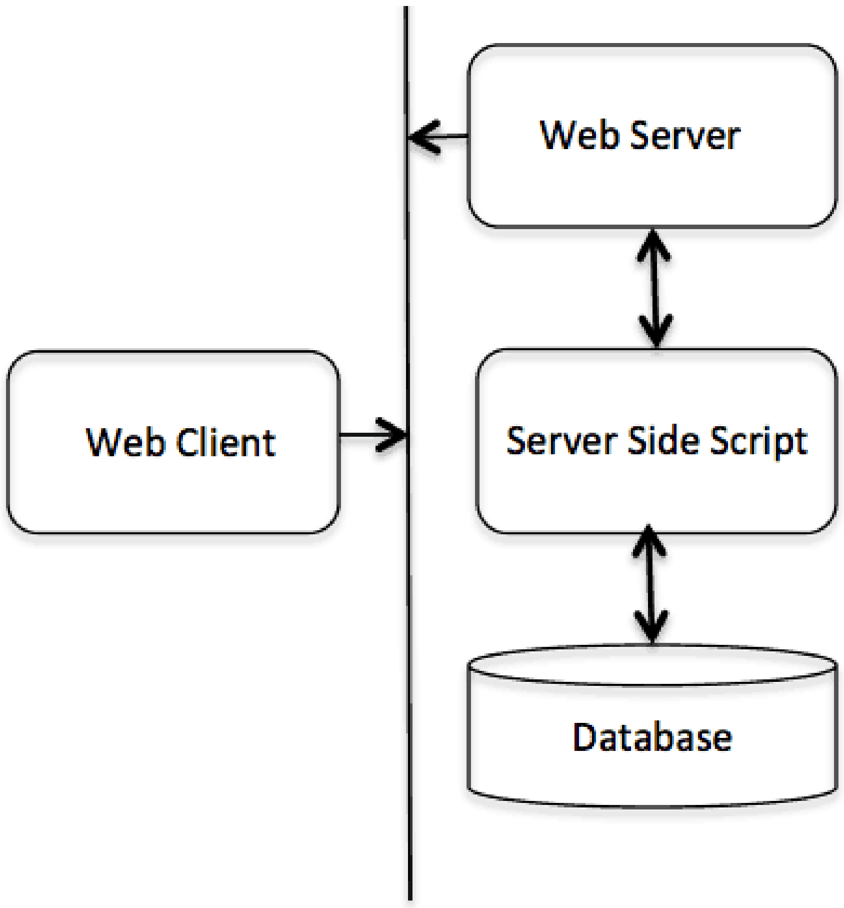
- HTTP is connectionless: The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client disconnects from the server and waits for a response. The server processes the request and re-establishes the connection with the client to send a response back
- HTTP is media 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:



HTTP Protocol

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.

History

The term hypertext was coined by Ted Nelson in 1965 in the Xanadu Project, which was in turn inspired by Vannevar Bush's vision (1930s) . Tim Berners-Lee and his team are credited with inventing the original HTTP along with HTML and the associated technology for a web server and a text-based web browser. Berners-Lee first proposed the "World Wide Web" project in 1989 . 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.

The first documented version of HTTP was HTTP V0.9 (1991). Dave Raggett led the HTTP Working Group in 1995 and wanted to expand the protocol with extended operations, extended negotiation, richer meta-information, tied with a security protocol which became more efficient by adding additional methods and header fields. RFC 1945 officially introduced and recognized HTTP V1.0 in 1996.

The HTTP WG planned to publish new standards in December 1995 and the support for pre-standard HTTP/1.1 based on the then developing RFC 2068 (called HTTP-NG) was rapidly adopted by the major browser developers.

By March 1996, pre-standard HTTP/1.1 was supported in (Arena, Netscape 2.0, Netscape Navigator Gold , Mosaic 2.7,and in Internet Explorer 2.0). End-user adoption of the new browsers was rapid.

In March 1996, one web hosting company reported that over 40% of browsers in use on the Internet were HTTP 1.1 compliant.That same web hosting company reported that by June 1996, 65% of all browsers accessing their servers were HTTP/1.1 compliant. The HTTP/1.1 standard as defined in RFC 2068 was officially released in January 1997. Improvements and updates to the HTTP/1.1 standard were released under RFC 2616 in 1999.

Tim Berners Lee

In 2007, the HTTP WG is Working Group was formed, in part, to revise and clarify the HTTP/1.1 specification. In June 2014, the WG released an updated six-part specification obsoleting RFC 2616:

Refernces

- RFC 7230, Message Syntax and Routing

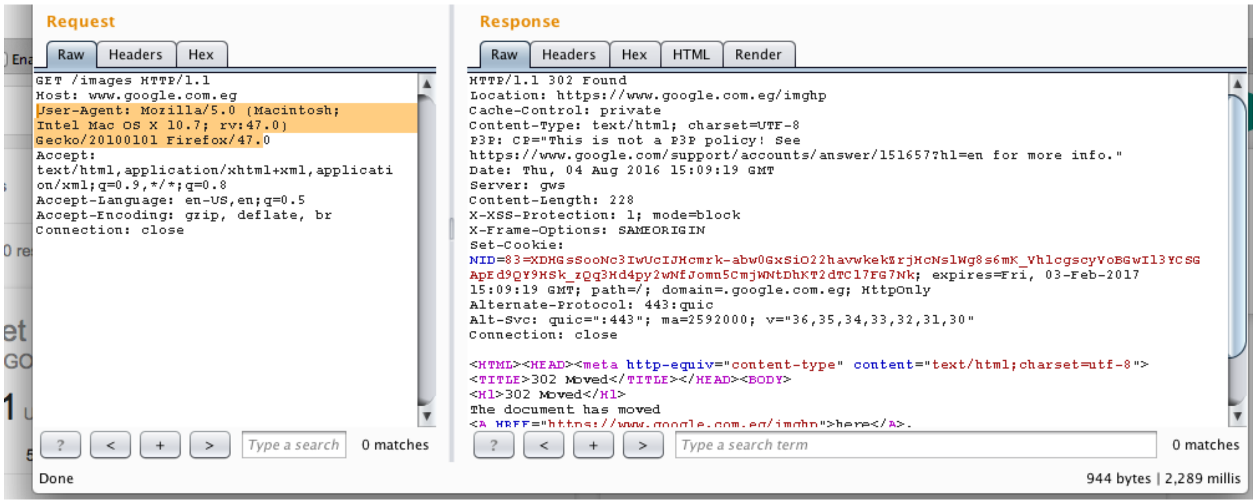
- RFC 7231, Semantics and Content
- RFC 7232, Conditional Requests
- RFC 7233, Range Requests
- RFC 7234, Caching
- RFC 7235, Authentication
- HTTP/2 was published as RFC 7540 in May 2015.

Request methods

HTTP defines methods (sometimes referred to as verbs) to indicate the desired action to be performed on the identified resource. What this resource represents, whether pre-existing data or data that is generated dynamically, depends on the implementation of the server. Often, the resource corresponds to a file or the output of an executable residing on the server. The HTTP/1.0 specification defined the GET, POST and HEAD methods and the HTTP/1.1 specification[12] added 5 new methods: OPTIONS, PUT, DELETE, TRACE and CONNECT. By being specified in these documents their semantics are well known and can be depended on. Any client can use any method and the server can be configured to support any combination of methods. If a method is unknown to an intermediate it will be treated as an unsafe and non-idempotent method. There is no limit to the number of methods that can be defined and this allows for future methods to be specified without breaking existing infrastructure. For example, WebDAV defined 7 new methods and RFC 5789 specified the PATCH method.

Example

An HTTP 1.1 request made using Burp suite . The request message, response header section, and response body as in the image.



```
GET /images HTTP/1.1
Host: www.google.com.eg
User-Agent: Mozilla/5.0 (Macintosh;
Intel Mac OS X 10.7; rv:47.0)
Gecko/20100101 Firefox/47.0
Accept:
text/html,application/xhtml+xml,applicati
on/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: close
```

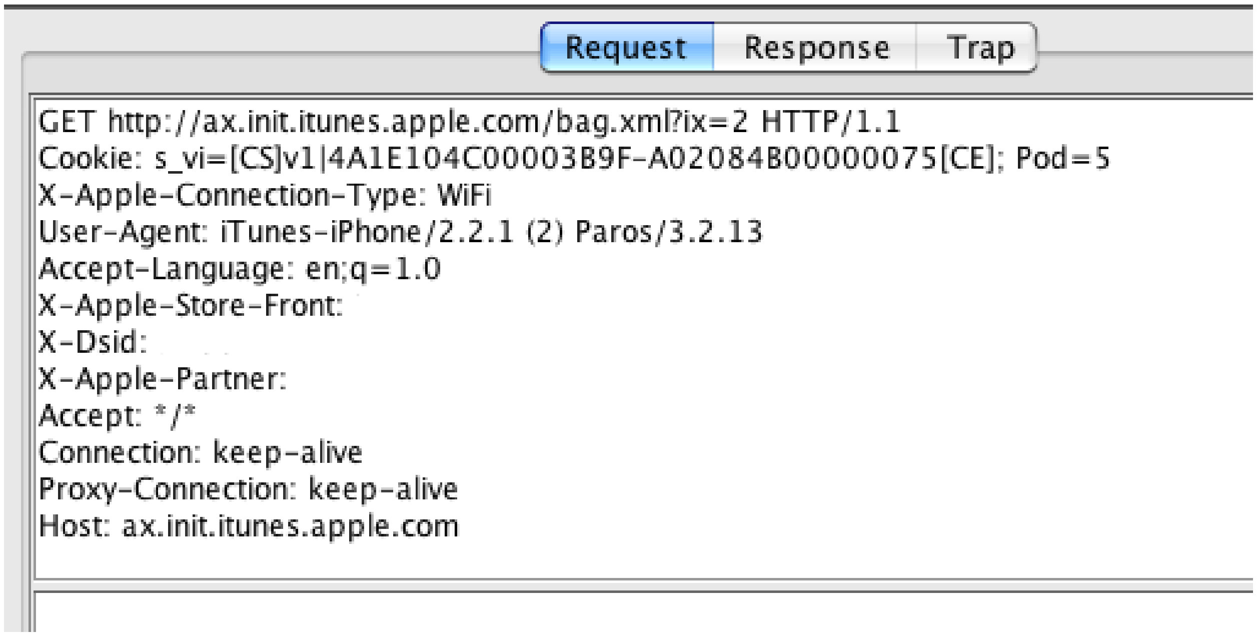
```
HTTP/1.1 302 Found
Location: https://www.google.com.eg/imghp
Cache-Control: private
Content-Type: text/html; charset=UTF-8
P3P: CP="This is not a P3P policy! See
https://www.google.com/support/accounts/answer/151657?hl=en for more info."
Date: Thu, 04 Aug 2016 15:09:19 GMT
Server: gws
Content-Length: 228
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Set-Cookie:
NID=83=XDHGsSooNc3IwUcIJHcmrk-abw0GxSiO22havwkek8rjHcNslWg8s6mK_VhlcgscyVoBGwIl3YCSG
ApEd9QY9HSk_zQq3Hd4py2wNfJomn5CmjWNtDhKT2dTCl7FG7Nk; expires=Fri, 03-Feb-2017
15:09:19 GMT; path=/; domain=.google.com.eg; HttpOnly
Alternate-Protocol: 443:quic
Alt-Svc: quic=":443"; ma=2592000; v="36,35,34,33,32,31,30"
Connection: close

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="https://www.google.com.eg/imghp">here</A>.
```

Get Request with burp suite

GET

The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect. (This is also true of some other HTTP methods.) The W3C has published guidance principles on this distinction, saying, **Web application design should be informed by the above principles, but also by the relevant limitations**.

The most common verbose is GET.



```
GET http://ax.init.itunes.apple.com/bag.xml?ix=2 HTTP/1.1
Cookie: s_vi=[CS]v1|4A1E104C00003B9F-A02084B00000075[CE]; Pod=5
X-Apple-Connection-Type: WiFi
User-Agent: iTunes-iPhone/2.2.1 (2) Paros/3.2.13
Accept-Language: en;q=1.0
X-Apple-Store-Front:
X-Dsid:
X-Apple-Partner:
Accept: */*
Connection: keep-alive
Proxy-Connection: keep-alive
Host: ax.init.itunes.apple.com
```

Get Request

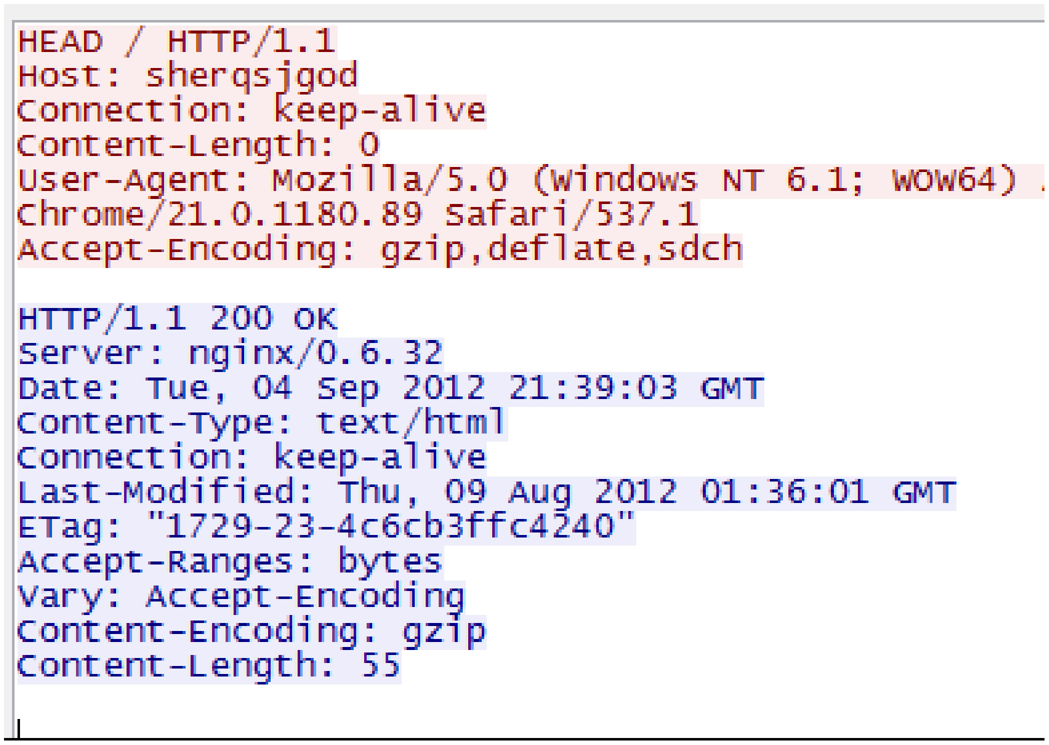
```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

The server response against the above GET request will be as follows:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Vary: Authorization,Accept
Accept-Ranges: bytes
Content-Length: 88
Content-Type: text/html
Connection: Closed

<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

HEAD

The HEAD method asks for a response identical to that of a GET request, but without the response body. This is useful for retrieving meta-information written in response headers, without having to transport the entire content.



```
HEAD / HTTP/1.1
Host: sherqsjgod
Connection: keep-alive
Content-Length: 0
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64)
Chrome/21.0.1180.89 Safari/537.1
Accept-Encoding: gzip,deflate,sdch

HTTP/1.1 200 OK
Server: nginx/0.6.32
Date: Tue, 04 Sep 2012 21:39:03 GMT
Content-Type: text/html
Connection: keep-alive
Last-Modified: Thu, 09 Aug 2012 01:36:01 GMT
ETag: "1729-23-4c6cb3ffc4240"
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 55
```

Head Request

The following example makes use of HEAD method to fetch header information about hello.htm:

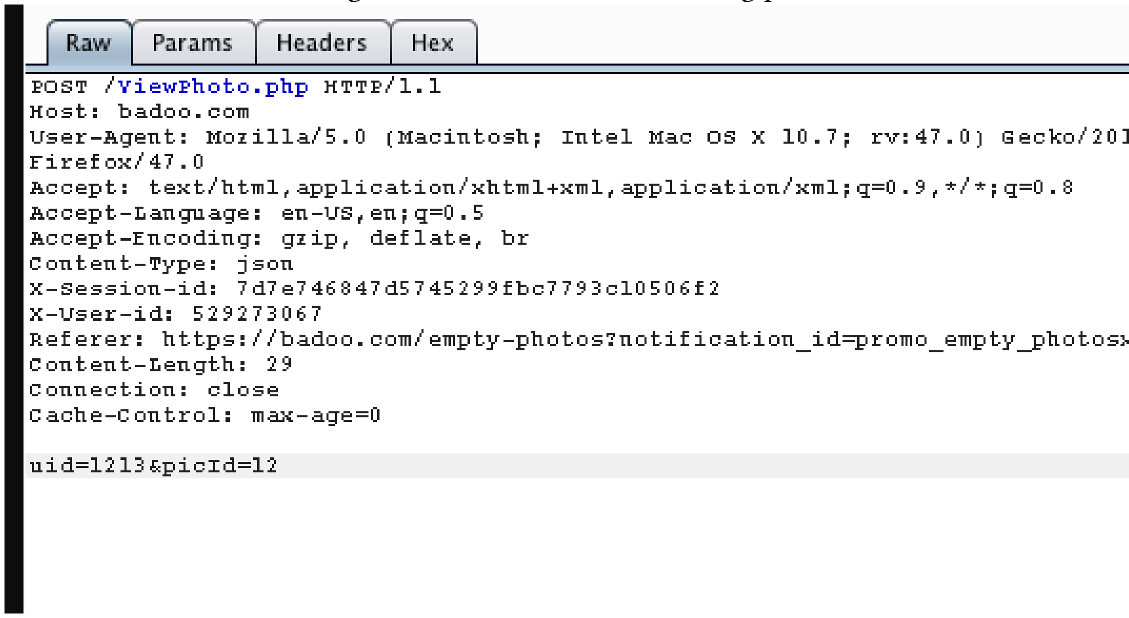
```
HEAD /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

The server response against the above GET request will be as follows:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Vary: Authorization,Accept
Accept-Ranges: bytes
Content-Length: 88
Content-Type: text/html
Connection: Closed
```

POST

The POST method requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, for example, an annotation for existing resources; a message for a bulletin board, newsgroup, mailing list, or comment thread; a block of data that is the result of submitting a web form to a data-handling process; or an



```
POST /ViewPhoto.php HTTP/1.1
Host: badoo.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:47.0) Gecko/201
Firefox/47.0
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, br
Content-Type: json
X-Session-id: 7d7e746847d5745299fbc7793c10506f2
X-User-id: 529273067
Referer: https://badoo.com/empty-photos?notification_id=promo_empty_photos
Content-Length: 29
Connection: close
Cache-Control: max-age=0

uid=1213&picId=12
```

item to add to a database.

The following example makes use of POST method to send a form data to the server, which will be processed by a process.cgi and finally a response will be returned:

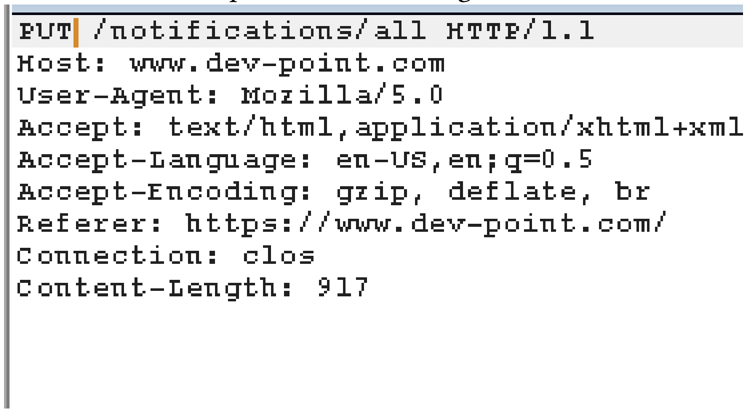
```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: text/xml; charset=utf-8
Content-Length: 88
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://clearforest.com/">string</string>
```

The server side script process.cgi processes the passed data and sends the following response:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Vary: Authorization,Accept
Accept-Ranges: bytes
Content-Length: 88
Content-Type: text/html
Connection: Closed

<html>
<body>
<h1>Request Processed Successfully</h1>
</body>
</html>
```

PUT

The PUT method requests that the enclosed entity be stored under the supplied URI. If the URI refers to an already existing resource, it is modified; if the URI does not point to an existing resource, then



```
PUT /notifications/all HTTP/1.1
Host: www.dev-point.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://www.dev-point.com/
Connection: clos
Content-Length: 917
```

the server can create the resource with that URI.

The following example requests the server to save the given entity-boy in hello.htm at the root of the server:

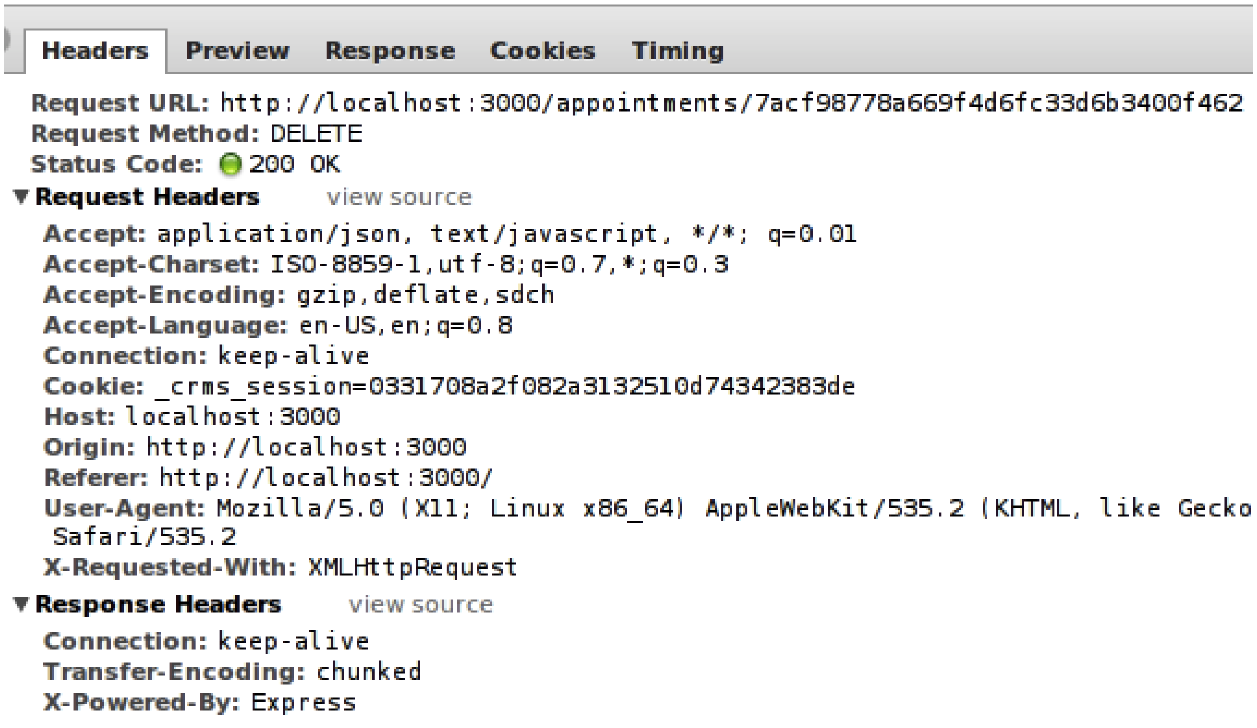
```
PUT /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Connection: Keep-Alive
Content-type: text/html
Content-Length: 182

<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

The server will store the given entity-body in hello.htm file and will send the following response back to the client:

```
HTTP/1.1 201 Created
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-type: text/html
Content-length: 30
Connection: Closed

<html>
<body>
<h1>The file was created.</h1>
</body>
</html>
```

DELETE

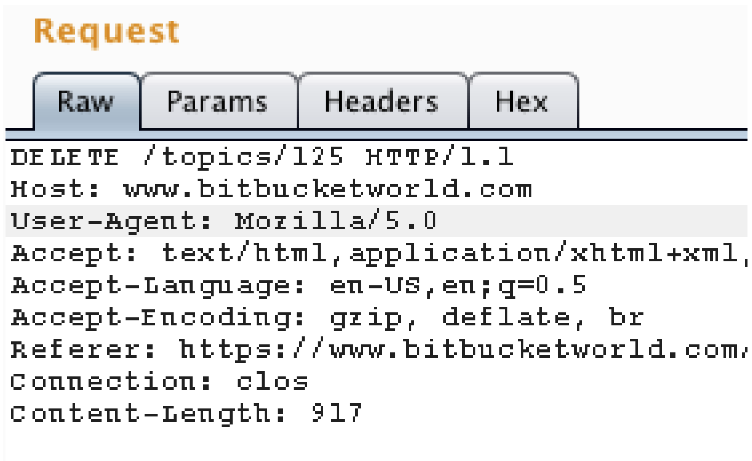
The DELETE method deletes the specified resource. Example From local host



```
Request URL: http://localhost:3000/appointments/7acf98778a669f4d6fc33d6b3400f462
Request Method: DELETE
Status Code: 200 OK
▼Request Headers     view source
  Accept: application/json, text/javascript, */*; q=0.01
  Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
  Accept-Encoding: gzip,deflate,sdch
  Accept-Language: en-US,en;q=0.8
  Connection: keep-alive
  Cookie: _crms_session=0331708a2f082a3132510d74342383de
  Host: localhost:3000
  Origin: http://localhost:3000
  Referer: http://localhost:3000/
  User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/535.2 (KHTML, like Gecko
   Safari/535.2
  X-Requested-With: XMLHttpRequest
▼Response Headers     view source
  Connection: keep-alive
  Transfer-Encoding: chunked
  X-Powered-By: Express
```

Delete Example

Another example from BBWorld website that deletes a request .



```
DELETE /topics/125 HTTP/1.1
Host: www.bitbucketworld.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://www.bitbucketworld.com/
Connection: clos
Content-Length: 917
```

Another example

The following example requests the server to delete the given file hello.htm at the root of the server:

```
DELETE /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Connection: Keep-Alive
```

The server will delete the mentioned file hello.htm and will send the following response back to the client:

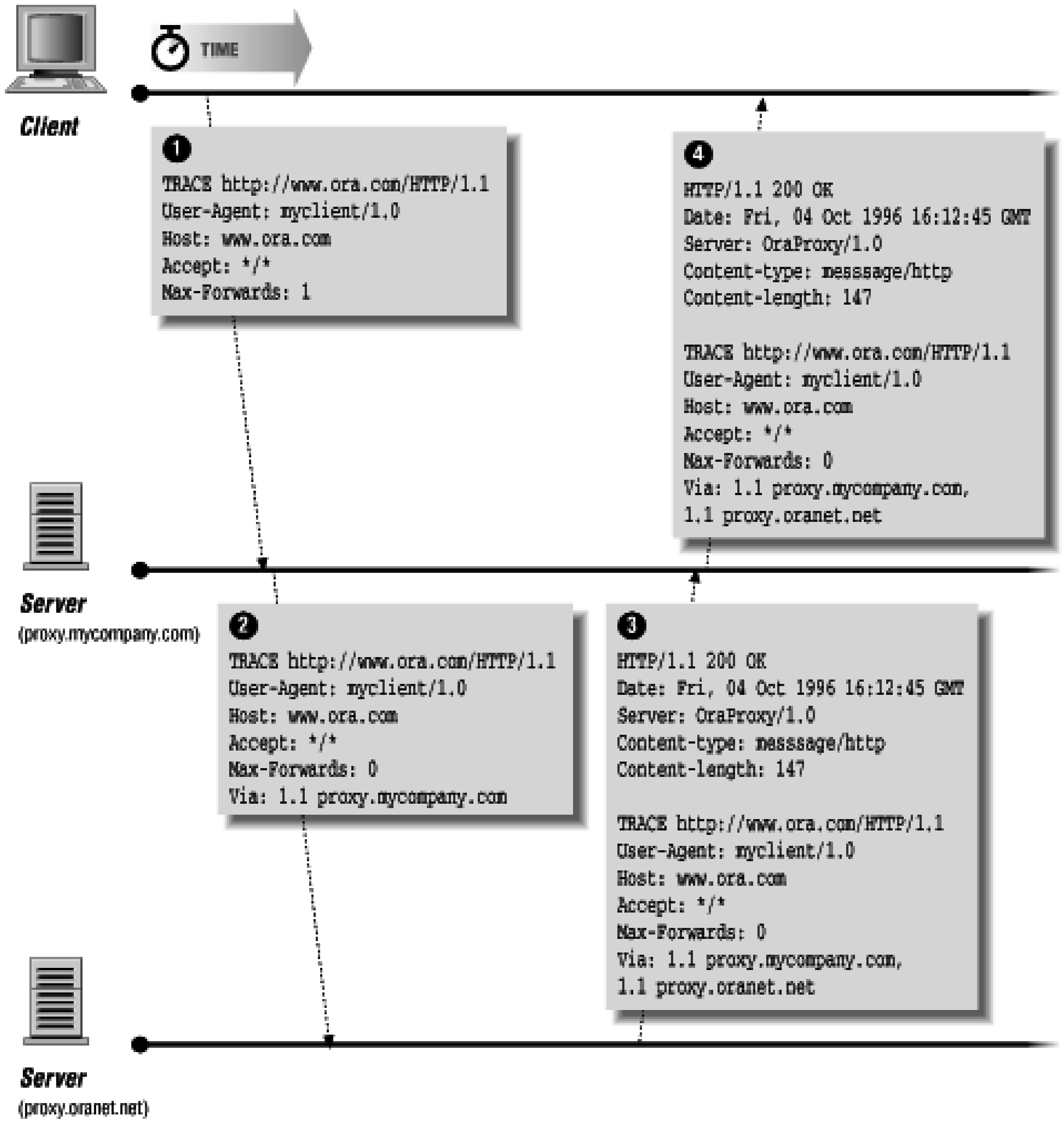
```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-type: text/html
Content-length: 30
Connection: Closed

<html>
<body>
<h1>URL deleted.</h1>
</body>
</html>
```

TRACE

The TRACE method echoes the received request so that a client can see what (if any) changes or additions have been made by intermediate servers.

Example of Trace Method

Another example



```
TRACE http://www.ora.com/HTTP/1.1
User-Agent: myclient/1.0
Host: www.ora.com
Accept: */*
Max-Forwards: 1
```

```
TRACE http://www.ora.com/HTTP/1.1
User-Agent: myclient/1.0
Host: www.ora.com
Accept: */*
Max-Forwards: 0
Via: 1.1 proxy.mycompany.com
```

```
HTTP/1.1 200 OK
Date: Fri, 04 Oct 1996 16:12:45 GMT
Server: OraProxy/1.0
Content-type: messsage/http
Content-length: 147

TRACE http://www.ora.com/HTTP/1.1
User-Agent: myclient/1.0
Host: www.ora.com
Accept: */*
Max-Forwards: 0
Via: 1.1 proxy.mycompany.com,
1.1 proxy.oranet.net
```

```
HTTP/1.1 200 OK
Date: Fri, 04 Oct 1996 16:12:45 GMT
Server: OraProxy/1.0
Content-type: messsage/http
Content-length: 147

TRACE http://www.ora.com/HTTP/1.1
User-Agent: myclient/1.0
Host: www.ora.com
Accept: */*
Max-Forwards: 0
Via: 1.1 proxy.mycompany.com,
1.1 proxy.oranet.net
```

Trace 2

The following example shows the usage of TRACE method:

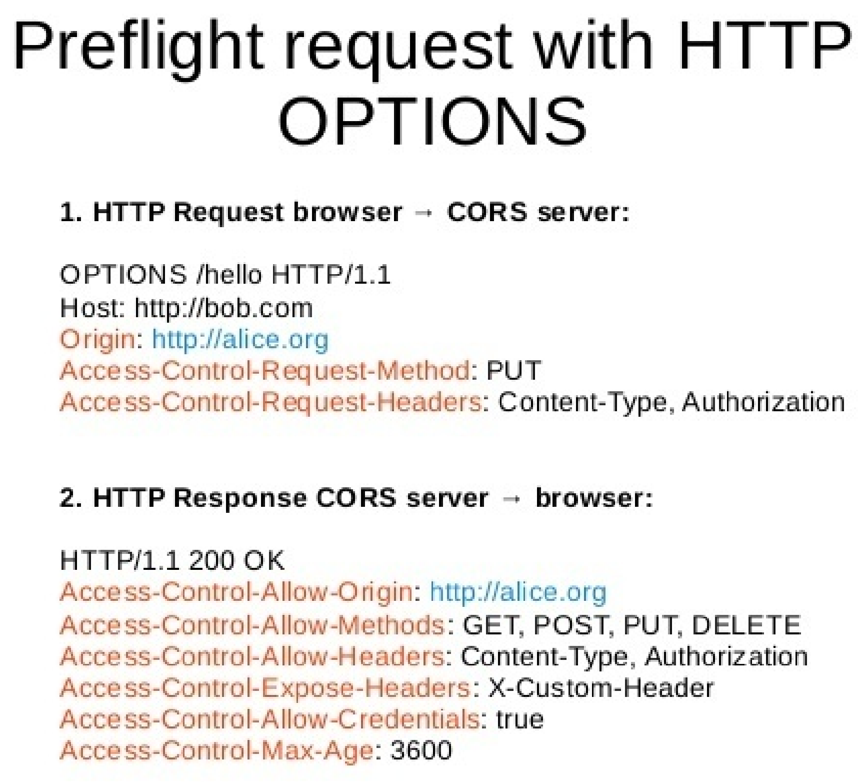
```
TRACE / HTTP/1.1
Host: www.tutorialspoint.com
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

The server will send the following message in response to the above request:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Connection: close
Content-Type: message/http
Content-Length: 39

TRACE / HTTP/1.1
Host: www.tutorialspoint.com
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

OPTIONS

The OPTIONS method returns the HTTP methods that the server supports for the specified URL. This can be used to check the functionality of a web server by requesting '*' instead of a specific resource.



Preflight request with HTTP OPTIONS

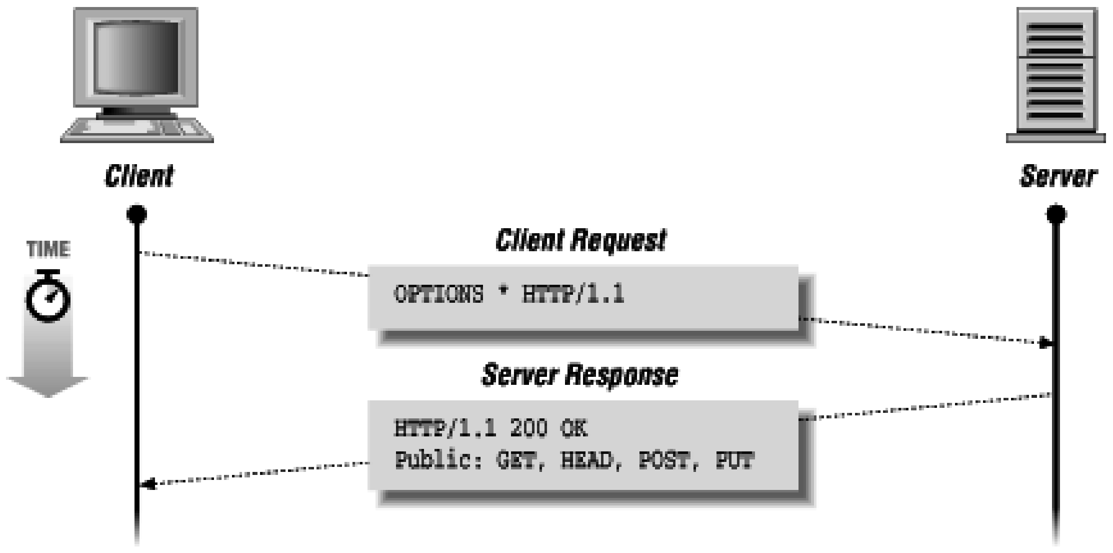
1. HTTP Request browser → CORS server:

```
OPTIONS /hello HTTP/1.1
Host: http://bob.com
Origin: http://alice.org
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: Content-Type, Authorization
```

2. HTTP Response CORS server → browser:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://alice.org
Access-Control-Allow-Methods: GET, POST, PUT, DELETE
Access-Control-Allow-Headers: Content-Type, Authorization
Access-Control-Expose-Headers: X-Custom-Header
Access-Control-Allow-Credentials: true
Access-Control-Max-Age: 3600
```

Options Request and response



Options Request and Response summary

The following example requests a connection with a web server running on the host tutorialspoint.com:

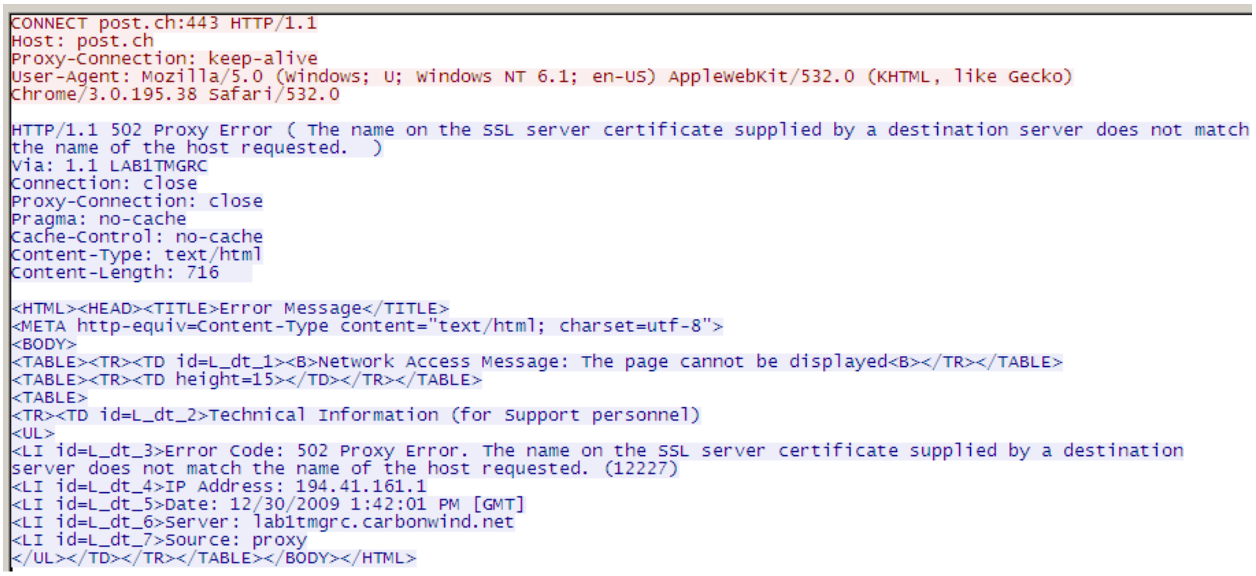
```
CONNECT www.tutorialspoint.com HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

The connection is established with the server and the following response is sent back to the client:

```
HTTP/1.1 200 Connection established
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
```

CONNECT

The CONNECT method converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communication (HTTPS) through an unencrypted HTTP proxy.



```
CONNECT post.ch:443 HTTP/1.1
Host: post.ch
Proxy-Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/532.0 (KHTML, like Gecko)
Chrome/3.0.195.38 Safari/532.0

HTTP/1.1 502 Proxy Error ( The name on the SSL server certificate supplied by a destination server does not match
the name of the host requested.  )
Via: 1.1 LAB1TMGRC
Connection: close
Proxy-Connection: close
Pragma: no-cache
Cache-Control: no-cache
Content-Type: text/html
Content-Length: 716

<HTML><HEAD><TITLE>Error Message</TITLE>
<META http-equiv=Content-Type content="text/html; charset=utf-8">
<BODY>
<TABLE><TR><TD id=L_dt_1><B>Network Access Message: The page cannot be displayed<B></TR></TABLE>
<TABLE><TR><TD height=15></TD></TR></TABLE>
<TABLE>
<TR><TD id=L_dt_2>Technical Information (for support personnel)
<UL>
<LI id=L_dt_3>Error Code: 502 Proxy Error. The name on the SSL server certificate supplied by a destination
server does not match the name of the host requested. (12227)
<LI id=L_dt_4>IP Address: 194.41.161.1
<LI id=L_dt_5>Date: 12/30/2009 1:42:01 PM [GMT]
<LI id=L_dt_6>Server: lab1tmgrc.carbonwind.net
<LI id=L_dt_7>Source: proxy
</UL></TD></TR></TABLE></BODY></HTML>
```

Connect Request and response

The following example requests a list of methods supported by a web server running on tutorialspoint.com:

```
OPTIONS * HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

The server will send an information based on the current configuration of the server, for example:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Allow: GET,HEAD,POST,OPTIONS,TRACE
Content-Type: httpd/unix-directory
```

PATCH

The PATCH method applies partial modifications to a resource.All general-purpose HTTP servers are required to implement at least the GET and HEAD methods, and, whenever possible, also the OPTIONS method.

Safe methods

Some of the methods (for example, HEAD, GET, OPTIONS) are, by convention, defined as safe, which means they are intended only for information retrieval and should not change the state of the server. In other words, they should not have side effects, beyond relatively harmless effects such as logging, caching, the serving of banner advertisements or incrementing a web counter. Making arbitrary GET requests without regard to the context of the application's state should therefore be considered safe. However, this is not mandated by the standard, and it is explicitly acknowledged that it cannot be guaranteed.

By contrast, methods such as POST, PUT, DELETE and PATCH are intended for actions that may cause side effects either on the server, or external side effects such as financial transactions or transmission of email. Such methods are therefore not usually used by conforming web robots or web crawlers; some that do not conform tend to make requests without regard to context or consequences.

Despite the prescribed safety of GET requests, in practice their handling by the server is not technically limited in any way. Therefore, careless or deliberate programming can cause non-trivial changes on the server. This is discouraged, because it can cause problems for web caching, search engines and other automated agents, which can make unintended changes on the server.

Idempotent methods and web applications Methods PUT and DELETE are defined to be idempotent, meaning that multiple identical requests should have the same effect as a single request (note that idempotence refers to the state of the system after the request has completed, so while the action the server takes (e.g. deleting a record) or the response code it returns may be different on subsequent requests, the system state will be the same every time). Methods GET, HEAD, OPTIONS and TRACE, being prescribed as safe, should also be idempotent, as HTTP is a stateless protocol.

In contrast, the POST method is not necessarily idempotent, and therefore sending an identical POST request multiple times may further affect state or cause further side effects (such as financial transactions). In some cases this may be desirable, but in other cases this could be due to an accident, such as when a user does not realize that their action will result in sending another request, or they did not receive adequate feedback that their first request was successful. While web browsers may show alert dialog boxes to warn users in some cases where reloading a page may re-submit a POST request, it is generally up to the web application to handle cases where a POST request should not be submitted more than once.

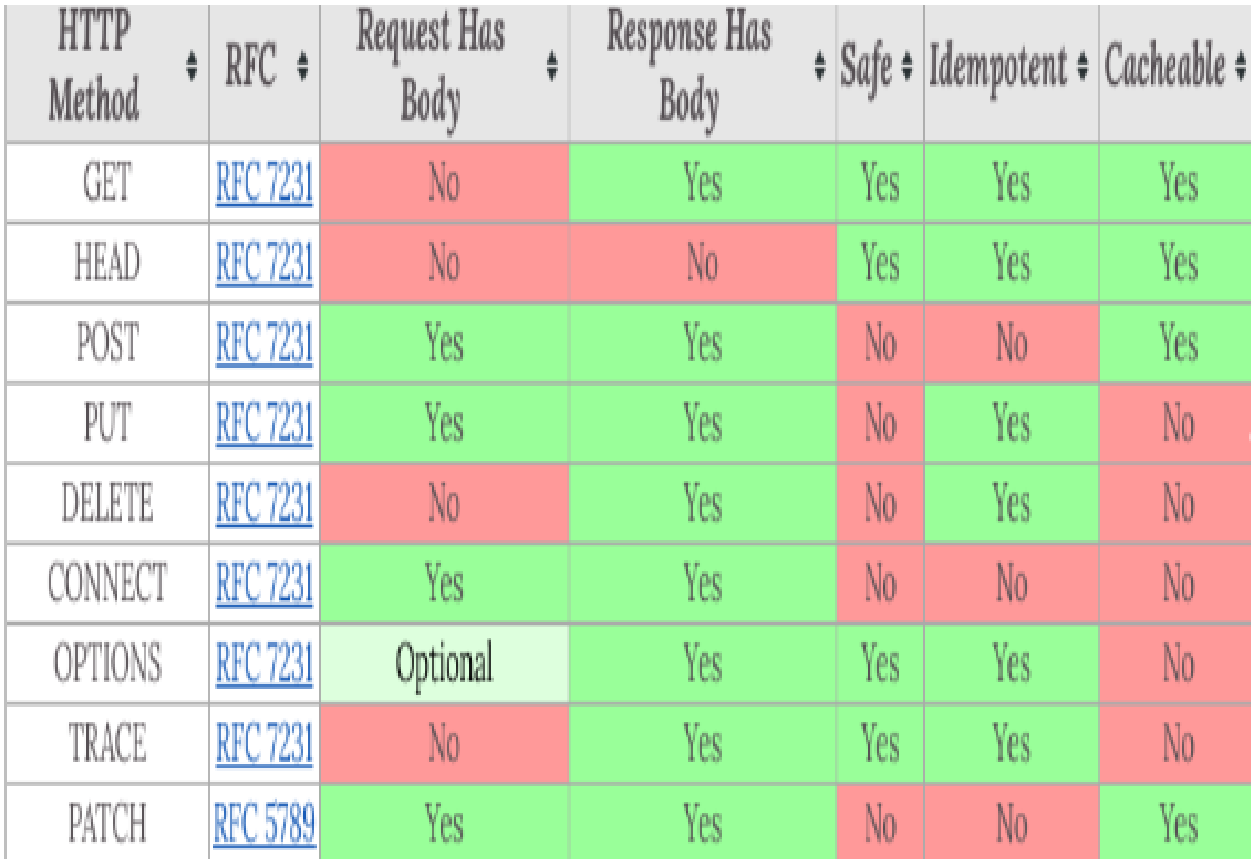
Note that whether a method is idempotent is not enforced by the protocol or web server. It is perfectly possible to write a web application in which (for example) a database insert or other non-idempotent action is triggered by a GET or other request. Ignoring this recommendation, however, may result in undesirable consequences, if a user agent assumes that repeating the same request is safe when it isn't.

Security

The TRACE method can be used as part of a class of attacks known as cross-site tracing; for that reason, common security advice is for it to be disabled in the server configuration. Microsoft IIS supports a proprietary "TRACK" method, which behaves similarly, and which is likewise recommended to be disabled.

Summary table

you can look this table to know the differnce between each method.



HTTP Method	RFC	Request Has Body	Response Has Body	Safe	Idempotent	Cacheable
GET	RFC 7231	No	Yes	Yes	Yes	Yes
HEAD	RFC 7231	No	No	Yes	Yes	Yes
POST	RFC 7231	Yes	Yes	No	No	Yes
PUT	RFC 7231	Yes	Yes	No	Yes	No
DELETE	RFC 7231	No	Yes	No	Yes	No
CONNECT	RFC 7231	Yes	Yes	No	No	No
OPTIONS	RFC 7231	Optional	Yes	Yes	Yes	No
TRACE	RFC 7231	No	Yes	Yes	Yes	No
PATCH	RFC 5789	Yes	Yes	No	No	Yes

Methods comparisment

Refernces

- RFC[1]
- HTTP Methods on Wikipedia[2]
- webDav[3]
- Patch[4]
- Web Application[5]

[1]https://tools.ietf.org/html/rfc5789
[2]https://www.wikiwand.com/en/Hypertext_Transfer_Protocol#Idempotent_methods_and_web_applications
[3]https://www.wikiwand.com/en/WebDAV
[4]https://www.wikiwand.com/en/Patch_verb
[5]https://www.wikiwand.com/en/Web_application

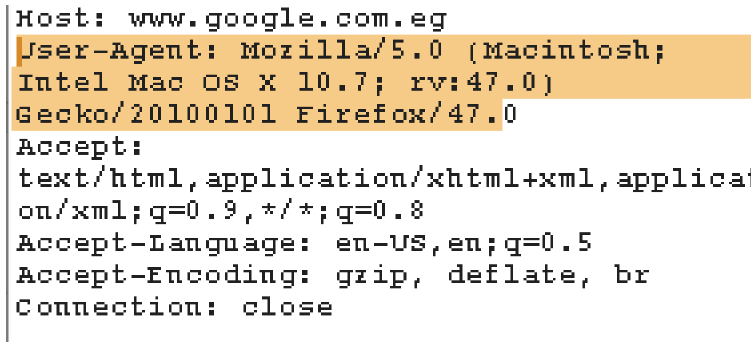
- Safe Methods[6]
- Post[7]
- Web resource[8]
- WebForm[9]
- URI[10]
- URL[11]
- TCP/IP Tunnel[12]
- Proxy[13]
- Side Effect[14]
- Caching[15]
- Server logs[16]
- Banner Advertisment[17]
- Web Counter[18]
- Robots[19]
- Web Crawler[20]
- Search Engines[21]
- Stateless Protocol[22]
- Alert Dialog box[23]
- User agent[24]
- XSC[25]
- Microsoft IIS[26]

[6]https://www.wikiwand.com/en/Hypertext_Transfer_Protocol#Safe_methods
[7]https://www.wikiwand.com/en/POST_(HTTP)
[8]https://www.wikiwand.com/en/Web_resource
[9]https://www.wikiwand.com/en/Form_(HTML)
[10]https://www.wikiwand.com/en/URI
[11]https://www.wikiwand.com/en/URL
[12]https://www.wikiwand.com/en/Tunneling_protocol
[13]https://www.wikiwand.com/en/HTTP_proxy
[14]https://www.wikiwand.com/en/Side_effect_(computer_science)
[15]https://www.wikiwand.com/en/Web_cache
[16]https://www.wikiwand.com/en/Server_log
[17]https://www.wikiwand.com/en/Web_banner
[18]https://www.wikiwand.com/en/Web_counter
[19]https://www.wikiwand.com/en/Internet_bot
[20]https://www.wikiwand.com/en/Web_crawler
[21]https://www.wikiwand.com/en/Search_engines
[22]https://www.wikiwand.com/en/Stateless_protocol
[23]https://www.wikiwand.com/en/Alert_dialog_box
[24]https://www.wikiwand.com/en/User_agent
[25]https://www.wikiwand.com/en/Cross-site_tracing
[26]https://www.wikiwand.com/en/Internet_Information_Services

Request message

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

- A request line (e.g., GET /images/logo.png HTTP/1.1, which requests a resource called /images/logo.png from the server).



```
Host: www.google.com.eg
User-Agent: Mozilla/5.0 (Macintosh;
Intel Mac OS X 10.7; rv:47.0)
Gecko/20100101 Firefox/47.0
Accept:
text/html,application/xhtml+xml,applicat
on/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: close
```

- Request header fields (e.g., Accept-Language: en).
- An empty line.
- An optional message body.
- The request line and other header fields must each end with <CR><LF> (that is, a carriage return character followed by a line feed character). The empty line must consist of only <CR><LF> and no other whitespace. In the HTTP/1.1 protocol, all header fields except Host are optional.

A request line containing only the path name is accepted by servers to maintain compatibility with HTTP clients before the HTTP/1.0 specification in RFC 1945.

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.

- 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.
- HEAD Same as GET, but it transfers the status line and the header section only.
- POST A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
- PUT Replaces all the current representations of the target resource with the uploaded content.
- DELETE Removes all the current representations of the target resource given by URI.
- CONNECT Establishes a tunnel to the server identified by a given URI.
- OPTIONS Describe the communication options for the target resource.
- 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
```

The asterisk The * 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
```

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
```

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 quick look

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.

Examples of Request Message

Now let's put it all together to form an HTTP request to fetch **hello.htm** page from the web server running on tutorialspoint.com

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Here we are not sending any request data to the server because we are fetching a plain HTML page from the server. Connection is a general-header, and the rest of the headers are request headers. The following example shows how to send form data to the server using request message body:

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

licenseID=string&content=string&/paramsXML=string
```

Here the given URL /cgi-bin/process.cgi will be used to process the passed data and accordingly, a response will be returned. Here **content-type** tells the server that the passed data is a simple web form data and **length** will be the actual length of the data put in the message body. The following example shows how you can pass plain XML to your web server:

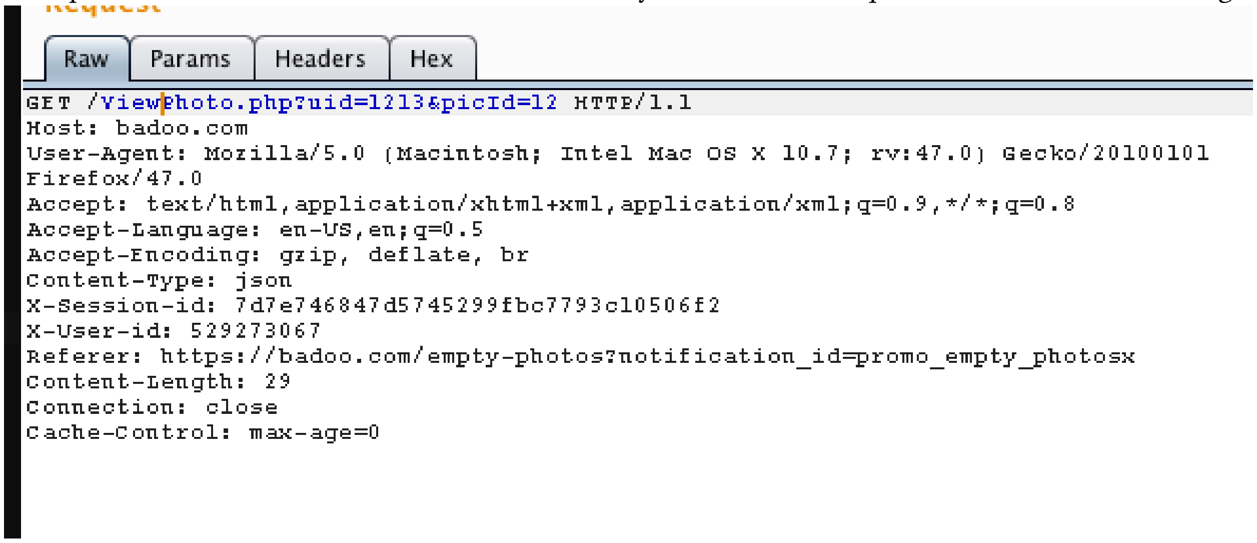
```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://clearforest.com/">string</string>
```

Request parametrs

There are two types of parametrs

1- URI Parametrs :

This parametrs are existed inside the URI , widely used in GET requests.Look at the next image.



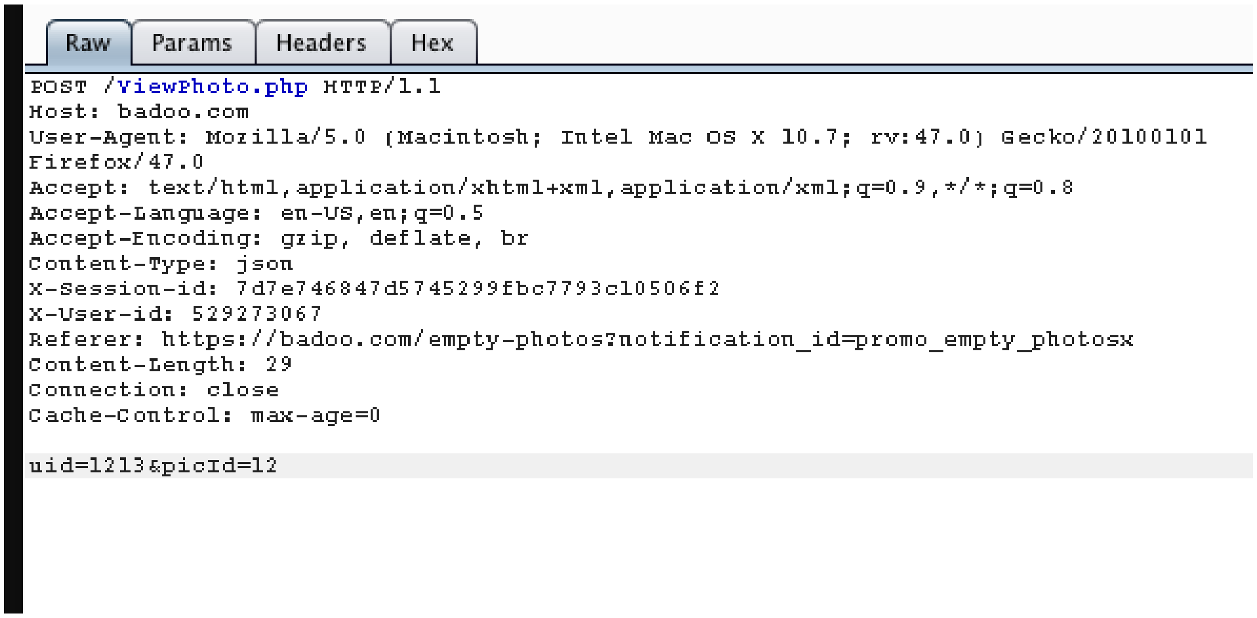
```
GET /ViewPhoto.php?uid=1213&picId=12 HTTP/1.1
Host: badoo.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:47.0) Gecko/20100101
Firefox/47.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: json
X-Session-id: 7d7e746847d5745299fbc7793c10506f2
X-User-id: 529273067
Referer: https://badoo.com/empty-photos?notification_id=promo_empty_photosx
Content-Length: 29
Connection: close
Cache-Control: max-age=0
```

- The requested URI is /ViewPhoto.php?uid=1213&picId=12
- and the requested page is viewPhoto.php

- after '?' this part is called queryString contain all parameters which separated by '&' -so we have two paramers are uid=1213&picId=12 , the first is uid (userId) which is equal to 1213,
- The second is picId (pictureID) which is equal to 12.

POST Parametrs

is the same as the previous one m but does not exist in the URI it exist on the body of the request so it is used in POST requests , see the same parameters with the same page but different method..



```
POST /ViewPhoto.php HTTP/1.1
Host: badoo.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:47.0) Gecko/20100101
Firefox/47.0
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, br
Content-Type: json
X-Session-id: 7d7e746847d5745299fbc7793c10506f2
X-User-id: 529273067
Referer: https://badoo.com/empty-photos?notification_id=promo_empty_photosx
Content-Length: 29
Connection: close
Cache-Control: max-age=0

uid=1213&picId=12
```

Post Parametrs

May a request contains both types of parametrs , this is legal.

POST parametrs can take any format (json , xml , any).

Parametrs

This chapter is going to list down few of the important HTTP Protocol Parameters and their syntax the way they are used in the communication. For example, format for date, format of URL, etc. This will help you in constructing your request and response messages while writing HTTP client or server programs. You will see the complete usage of these parameters in subsequent chapters while learning the message structure for HTTP requests and responses.

HTTP Version

HTTP uses a <major>.<minor> numbering scheme to indicate versions of the protocol. The version of an HTTP message is indicated by an HTTP-Version field in the first line. Here is the general syntax of specifying HTTP version number:

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

Example

```
HTTP/1.0

or

HTTP/1.1
```

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
```

Date/Time Formats

All HTTP date/time stamps MUST be represented in Greenwich Mean Time (GMT), without exception. HTTP applications are allowed to use any of the following three representations of date/time stamps:

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

Character Sets

We use character sets to specify the character sets that the client prefers. Multiple character sets can be listed separated by commas. If a value is not specified, the default is the US-ASCII.

Example Following are the valid character sets:

```
US-ASCII

or

ISO-8859-1

or

ISO-8859-7
```

Content Encodings

A content encoding value indicates that an encoding algorithm has been used to encode the content before passing it over the network. Content coding are primarily used to allow a document to be compressed or otherwise usefully transformed without losing the identity.

All content-coding values are case-insensitive. HTTP/1.1 uses content-coding values in the Accept-Encoding and Content-Encoding header fields which we will see in the subsequent chapters.

Example Following are the valid encoding schemes:

```
Accept-encoding: gzip

or

Accept-encoding: compress

or

Accept-encoding: deflate
```

Media Types

HTTP uses Internet Media Types in the Content-Type and Accept header fields in order to provide open and extensible data typing and type negotiation. All the Media-type values are registered with the Internet Assigned Number Authority (IANA). The general syntax to specify media type is as follows:

```
media-type     = type "/" subtype *( ";" parameter )
```

The type, subtype, and parameter attribute names are case–insensitive.

Example

```
Accept: image/gif
```

Language Tags

HTTP uses language tags within the Accept-Language and Content-Language fields. A language tag is composed of one or more parts: a primary language tag and a possibly empty series of subtags:

```
language-tag  = primary-tag *( "-" subtag )
```

White spaces are not allowed within the tag and all tags are case- insensitive.

Example Example tags include:

```
en, en-US, en-cockney, i-cherokee, x-pig-latin
```

where any two-letter primary-tag is an ISO-639 language abbreviation and any two-letter initial subtag is an ISO-3166 country code.

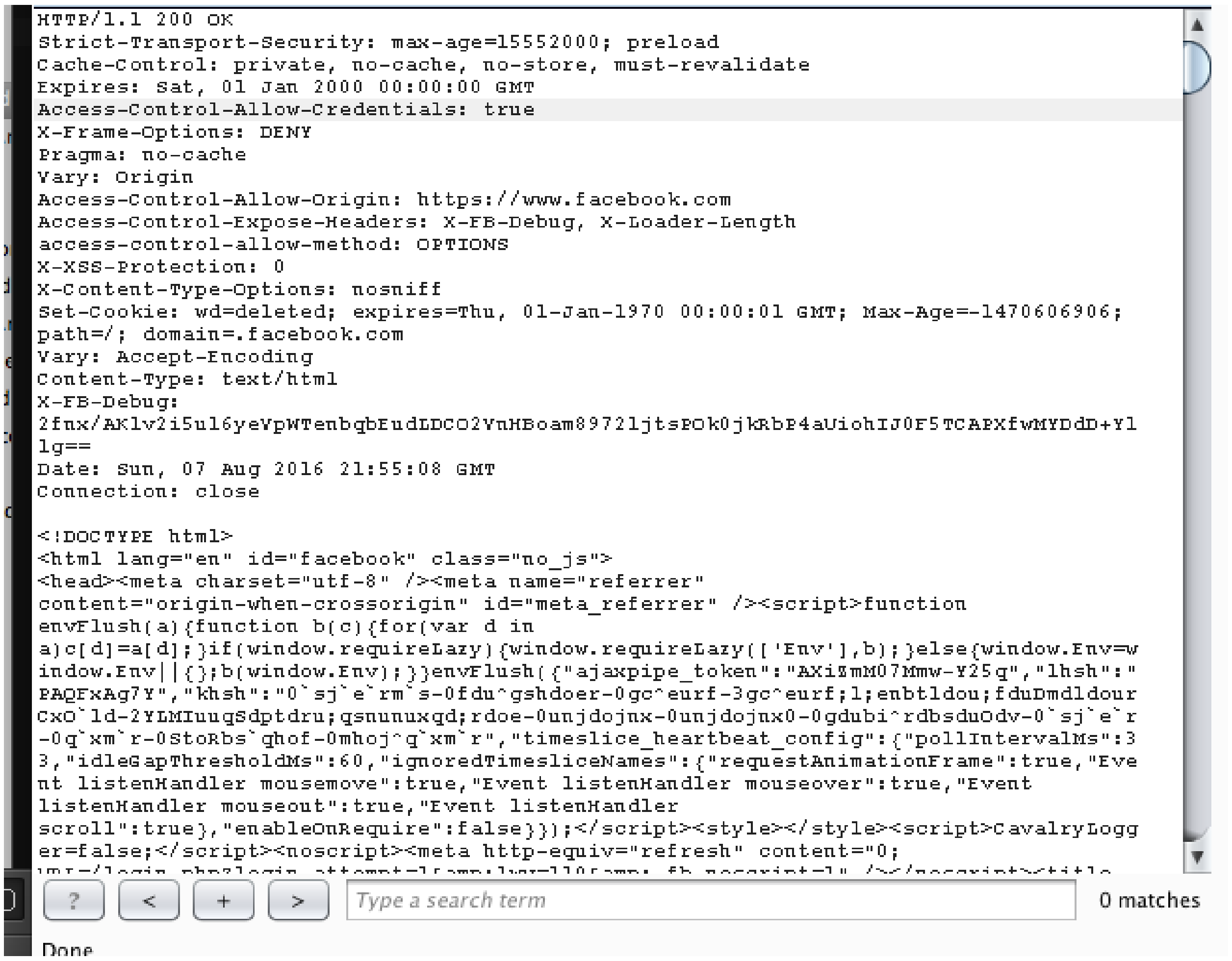
Response message

After receiving and interpreting a request message, a server responds with an HTTP response message.

The response message consists of the following:

- A status line which includes the status code and reason message (e.g., HTTP/1.1 200 OK, which indicates that the client's request succeeded).
- Response header fields (e.g., Content-Type: text/html).An empty line.
- An optional message body.

The status line and other header fields must all end with <CR><LF>. The empty line must consist of only <CR><LF> and no other whitespace. This strict requirement for <CR><LF> is relaxed somewhat within message bodies for consistent use of other system linebreaks such as <CR> or <LF> alone



```
HTTP/1.1 200 OK
Strict-Transport-Security: max-age=15552000; preload
Cache-Control: private, no-cache, no-store, must-revalidate
Expires: Sat, 01 Jan 2000 00:00:00 GMT
Access-Control-Allow-Credentials: true
X-Frame-Options: DENY
Pragma: no-cache
Vary: Origin
Access-Control-Allow-Origin: https://www.facebook.com
Access-Control-Expose-Headers: X-FB-Debug, X-Loader-Length
access-control-allow-method: OPTIONS
X-XSS-Protection: 0
X-Content-Type-Options: nosniff
Set-Cookie: wd=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=-1470606906;
path=/; domain=.facebook.com
Vary: Accept-Encoding
Content-Type: text/html
X-FB-Debug:
2fnx/AKlv2i5ul6yeVpWTenbqbEudLDCO2VnHBoam8972ljtsPOk0jkRbP4aUiohIJ0F5TCAPXfwMYDdD+Yl
lg==
Date: Sun, 07 Aug 2016 21:55:08 GMT
Connection: close

<!DOCTYPE html>
<html lang="en" id="facebook" class="no_js">
<head><meta charset="utf-8" /><meta name="referrer"
content="origin-when-crossorigin" id="meta_referrer" /><script>function
envFlush(a){function b(c){for(var d in
a)c[d]=a[d];}if(window.requireLazy){window.requireLazy(['Env'],b);}else{window.Env=w
indow.Env||{};b(window.Env);}}envFlush({"ajaxpipe_token":"AXiZmM07Mmw-Y25q","lhsh":"
PAQFxAg7Y","khsh":"0`sj`e`rm`s-0fdu^gshdoer-0gc^eurf-3gc^eurf;1;enbtldou;fduDmdldour
CxO`ld-2YLMIuugSdptdru;qsnunuxqd;rdoe-0unjdojnx-0unjdojnx0-0gdubi^rdbsduOdv-0`sj`e`r
-0q`xm`r-0StoRbs`qhof-0mhoj^q`xm`r","timeslice_heartbeat_config":{"pollIntervalMs":3
3,"idleGapThresholdMs":60,"ignoredTimesliceNames":{"requestAnimationFrame":true,"Eve
nt listenHandler mousemove":true,"Event listenHandler mouseover":true,"Event
listenHandler mouseout":true,"Event listenHandler
scroll":true},"enableOnRequire":false}});</script><style></style><script>CavalryLogg
er=false;</script><noscript><meta http-equiv="refresh" content="0;
```

Complete response contains all above three parts

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 quick look

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:

1xx: Informational It means the request was received and the process is continuing.

2xx: Success It means the action was successfully received, understood, and accepted.

3xx: Redirection It means further action must be taken in order to complete the request.

4xx: Client Error It means the request contains incorrect syntax or cannot be fulfilled.

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 quick look

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.

Examples of Response Message

Now let's put it all together to form an HTTP response for a request to fetch the hello.htm page from the web server running on tutorialspoint.com

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed

<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

The following example shows an HTTP response message displaying error condition when the web server could not find the requested page:

```
HTTP/1.1 404 Not Found
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 230
Connection: Closed
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head>
   <title>404 Not Found</title>
</head>
```

```
<body>
   <h1>Not Found</h1>
   <p>The requested URL /t.html was not found on this server.</p>
</body>
</html>
```

Following is an example of HTTP response message showing error condition when the web server encountered a wrong HTTP version in the given HTTP request:

```
HTTP/1.1 400 Bad Request
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 230
Content-Type: text/html; charset=iso-8859-1
Connection: Closed

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head>
   <title>400 Bad Request</title>
</head>
<body>
   <h1>Bad Request</h1>
   <p>Your browser sent a request that this server could not understand.</p>
   <p>The request line contained invalid characters following the protocol strin\
g.</p>
</body>
</html>
```