

# HTTP - Request/ Response

# HTTP - Documentation

- HTTP/1.1 is defined by RFC2616 of the IETF
  - <https://tools.ietf.org/html/rfc2616>
  - This is THE document for all your questions about HTTP
  - Today we'll discuss topics in sections 4, 5, and 6
- RFC
  - Request For Comments
  - Submit an RFC for public discussion or to publish information
- IETF
  - Internet Engineering Task Force
  - Adopts some RFC's as Internet standards

# New Lines

- A new line character in an HTTP request/response must be:
  - `"\r\n"`
    - Carriage return (From the days of typewriters)
    - New line
  - In the documentation this is referred to as a CRLF
  - CRLF == Carriage Return Line Feed
- Be aware of this while parsing
- Use `"\r\n"` for new lines when preparing your responses

# HTTP GET Request

# GET Request

- We'll use this simple request as an example

**GET / HTTP/1.1**  
**Host: cse312.com**

# GET Request

- More accurately, it will be this
- For the example, I will show "\r\n\r\n" as a new line
- Note that there is a blank line at the end of the request

**GET / HTTP/1.1\r\nHost: cse312.com\r\n\r\n**

# The Request Line

- The first line of the request is always the request line
- The request line has 3 values separated by spaces
  - The request type (GET/POST/etc)
  - The path of the request (ex. "/")
  - The HTTP Version
    - We'll always use HTTP/1.1 in this course
    - You can assume the request uses HTTP/1.1 in your assignments without checking this string

**GET / HTTP/1.1**

**Host: cse312.com**

# The Request Line

- Parse the request line by looking for the 2 space characters
  - Separate the values and check the strings
- Typically: When the root path "/" is requested, serve your home page
  - By convention, web servers look for index.html to serve
- If the url contains a different path, it will appear in the request line

**GET / HTTP/1.1**

**Host: cse312.com**

**GET /lecture HTTP/1.1**

**Host: cse312.com**



# Headers

- Following the request line are any number of headers
- HTTP Headers
  - Key-Value pairs
  - Key and value separated by a colon ":"
- Each header will be on a new line
- To parse, look for the colon ":" and read the key and value
  - There is an optional space after the colon which should be removed if present

**GET / HTTP/1.1**

**Host: cse312.com**

# HTTP Response

# Response

- Your web server will listen for HTTP requests over the TCP sockets and respond with HTTP responses
- Send this response back to the client to serve them the requested content

**HTTP/1.1 200 OK**

**Content-Type: text/plain**

**Content-Length: 5**

**hello**

# Response

- Or, more accurately

**HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\nContent-Length: 5\r\n\r\nhello**

# Status Line

- The first line of the response must be the status line
- Status line contains 3 values separated by spaces
  - The HTTP version
  - The status code
  - The status message (Reason phrase in docs)

**HTTP/1.1 200 OK**

**Content-Type: text/plain**

**Content-Length: 5**

**hello**

# Response Headers

- The headers in the response follow the same format as request headers
- Should have at least two headers
  - Content-Type - Tells the browser how to parse this content
  - Content-Length - How many bytes should be read from the body

**HTTP/1.1 200 OK**

**Content-Type: text/plain**

**Content-Length: 5**

**hello**

# Body

- The headers are followed by 2 new lines "\r\n\r\n" to indicate the beginning of the body
- The body contains the content that is being served

**HTTP/1.1 200 OK**  
**Content-Type: text/plain**  
**Content-Length: 5**

hello

# Response Codes

- Tells the browser the nature of the response
  - 200-level codes: Everything went well
  - 300-level codes: Redirect the request
  - 400-level codes: Error caused by the client
  - 500-level codes: Error caused by the server
- Include a human readable message

**HTTP/1.1 200 OK**

**Content-Type: text/plain**

**Content-Length: 5**

**hello**



# 404 Not Found

- If a path is requested that your server does not handle
  - Respond with a 404 Not Found
- The response format is the same as a 200 response
  - Include content type and length
  - Include a body that will be displayed to the client

**HTTP/1.1 404 Not Found**

**Content-Type: text/plain**

**Content-Length: 36**

**The requested content does not exist**

# 301 Moved Permanently

- Respond with 301 to redirect the user to a new path
  - Ex. When the server is updated with new paths, redirect the old paths to the new paths instead of maintaining both
  - Ex. Redirect HTTP requests to HTTPS requests

**HTTP/1.1 301 Moved Permanently**

**Content-Length: 0**

**Location: /new-path**

# 301 Moved Permanently

- A 301 response must contain a Location header
  - This is the path of the redirect
- The client will make a second request for the Location path

**HTTP/1.1 301 Moved Permanently**

**Content-Length: 0**

**Location: /new-path**

# 301 Moved Permanently

- If the Location is not a full url, it will be treated as a relative path
- New request is made with the same protocol/host/port as the original request
- Example:
  - First request was for "http://cse312.com:8080/old-path"
  - Second request is "http://cse312.com:8080/new-path"

**HTTP/1.1 301 Moved Permanently**

**Content-Length: 0**

**Location: /new-path**

# 301 Moved Permanently

- If the location is a full url, the user can be redirected to a different server
- Example:
  - First request was for "http://cse312.com:8080/old-path"
  - Second request is "https://google.com/"

**HTTP/1.1 301 Moved Permanently**

**Content-Length: 0**

**Location: <https://google.com/>**

# HTTP POST Request

# POST Request

- A POST request, or any request containing a body, will be formatted similar to your HTTP responses

```
POST /path HTTP/1.1  
Content-Type: text/plain  
Content-Length: 5
```

```
hello
```

# POST Request

- More accurately

**POST /path HTTP/1.1\r\nContent-Type: text/plain\r\nContent-Length: 5\r\n\r\nhello**



# POST Request

- When parsing, look for the Content-Length and Content-Type headers
- It is strongly recommended that you write general header parsing code that you can use for all requests

```
POST /path HTTP/1.1\r\nContent-Type: text/plain\r\nContent-Length: 5\r\n\r\nhello
```

# POST Request

- Look for the blank line that separates the headers from the body
  - "\r\n\r\n"
- Read everything after this blank line
- Make sure you've read "Content-Length" number of bytes
  - It's possible to only receive part of a request and have to read the rest from the TCP socket

**POST /path HTTP/1.1\r\nContent-Type: text/plain\r\nContent-Length: 5\r\n\r\nhello**

# POST Request

- When you read the content from the body:
  - Do whatever your server does based on it's feature for this path
  - Send a response to the client

**POST /path HTTP/1.1\r\nContent-Type: text/plain\r\nContent-Length: 5\r\n\r\nhello**

# Live Examples