HTTP: Request Messages

HTTP request messages are a pivotal part of the protocol. Let's have a close look at them!

WE'LL COVER THE FOLLOWING

Introduction

There are two types of HTTP messages as discussed previously:

- HTTP **request messages**
- HTTP **response messages**

We'll study request messages in this one.

HTTP Request Messages

Let's look at request messages first. Here is an example of a typical HTTP message:

```
GET /path/to/file/index.html HTTP/1.1
Host: www.educative.io
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
Accept: text/html
```

an HTTP request message

It should be noted that,

- HTTP messages are in **plain ASCII text**
- **Each line** of the message **ends with** two control characters: a **carriage return and a line feed**: `\r\n`.
    - The last line of the message also ends with a carriage return and a line feed!
- This particular message has 6 lines, but HTTP messages can have **one or as many lines as needed**.
- The first line is called the **request line** while the rest are called **header lines**.

The Anatomy of an HTTP Request Line

The HTTP request line is followed by an HTTP header. We'll look at the request line first. The request line consists of three parts:

- **Method**
- **URL**
- **Version**

Let's discuss each.

```
GET path/to/file HTTP 1.1
```

HTTP request line

1 of 4

Request Method

```
GET path/to/file HTTP 1.1
```

HTTP request method

2 of 4

URL to resource

3 of 4

HTTP Version

4 of 4

HTTP Methods

HTTP methods tell the server what to do. There are a lot of HTTP methods but we'll study the most common ones: `GET`, `POST`, `HEAD`, `PUT`, or `DELETE`.

- `GET` is the most common and **requests data**.
- `POST` **puts an object on the server**.
  - This method is generally used when the client is not sure where the new data would reside. The server responds with the location of the object.
  - The data posted can be a message for a bulletin board, newsgroup, mailing list, a command, a web form, or an item to add to a database.
  - The POST method technically requests a page but that depends on what was entered.
- `HEAD` is similar to the `GET` method except that **the resource requested does not get sent in response. Only the HTTP headers are sent** instead.
  - This is useful for quickly retrieving meta-information written in

response headers, without having to transport the entire content. In other words, it's useful to check with minimal traffic if a certain object still exists. This includes its meta-data, like the last modified date. The latter can be useful for caching.
    - This is also useful for testing and debugging.
- `PUT` **uploads an enclosed entity under a supplied URI**. In other words, it **puts** data at a specific location. If the URI refers to an already existing resource, it's replaced with the new one. If the URI does not point to an existing resource, then the server creates the resource with that URI.
- `DELETE` **deletes an object** at a given URL.

Note that while most forms are sent from the POST method, the GET method is also used sometimes with the entries of the form appended to the URL, as in arguments like this:

```
http://www.website.com/form.php/?Name=PostMan?Age=45?Interest=Post
```

forms with GET requests

However, sending forms with a POST request is generally better because:

1. The amount of data that can be sent via a post request is unlimited.
2. The form's fields are not shown in the URL.

> **Note: URIs & URLs**
>
> - **Uniform Resource Locators (URLs)** URLs are used to identify an object over the web. RFC 2396. A URL has the following format: `protocol://hostname:port/path-and-file-name`
> - **Uniform Resource Identifiers (URIs)** can be more specific than URLs in such a way that they can locate fragments within objects too RFC 3986. A URI has the following format: `http://host:port/path?request-parameters#nameAnchor` . For instance, `https://www.educative.io/collection/page/10370001/6105520698032128/6460983855808512/#http-methods` is a URI.

URL

This is the location that any HTTP method is referring to.

Version

The HTTP version is also specified in the request line. The latest version of HTTP is HTTP/2.

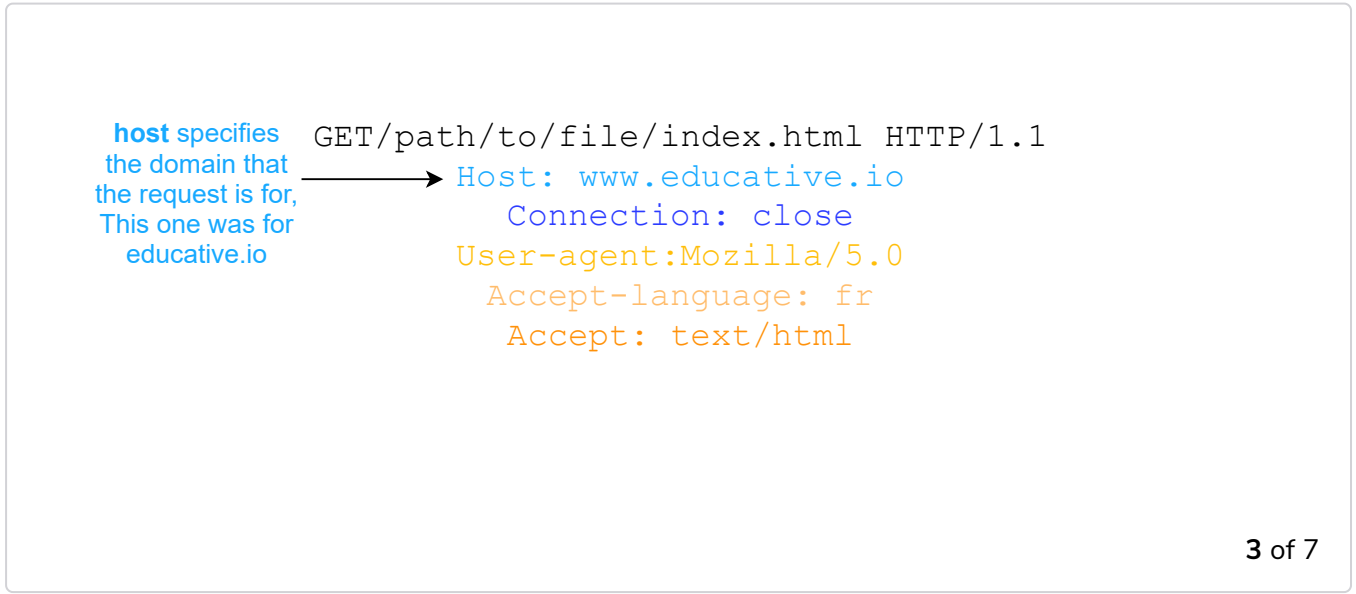
The Anatomy of HTTP Header Lines

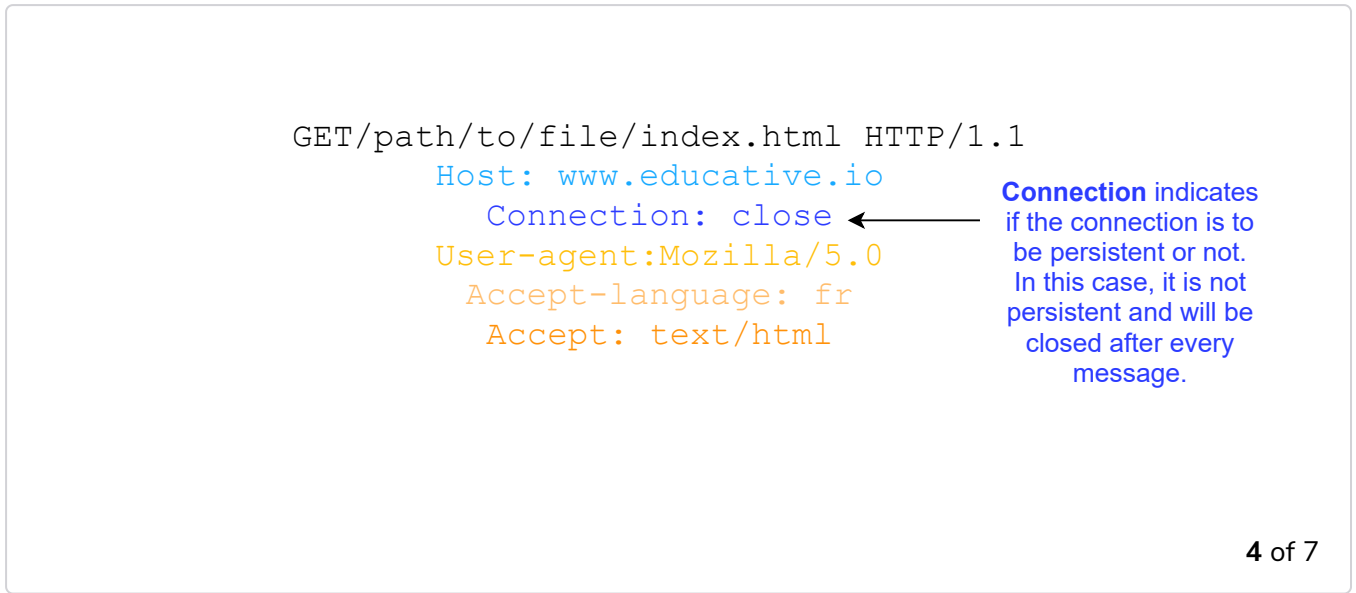
The HTTP request line is followed by an HTTP header. A lot of HTTP headers exist! We'll be covering the most important ones in this lesson. However, if you're interested, you can read further about all of them.

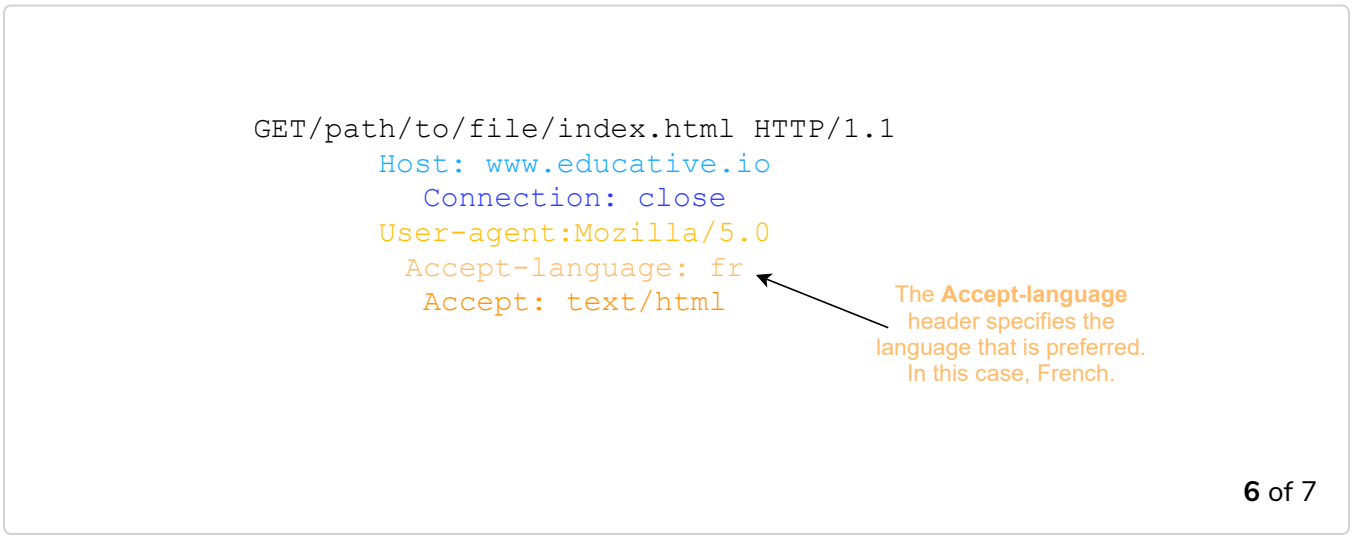
- The first header line specifies the `Host` that the request is for.
- The second one defines the type of HTTP `Connection`. It's Non-persistent in the case of the following drawing as the connection is specified to be closed.
- The `user-agent` line specifies the client. This is useful when the server has different web pages that exist for different devices and browsers.
- The `Accept-language` header specifies the language that is preferred. The server checks if a web page in that language exists and sends it if it does, otherwise the server sends the default page.
- The `Accept` header defines the sort of response to accept. It can be anything like HTML files, images, and audio/video.

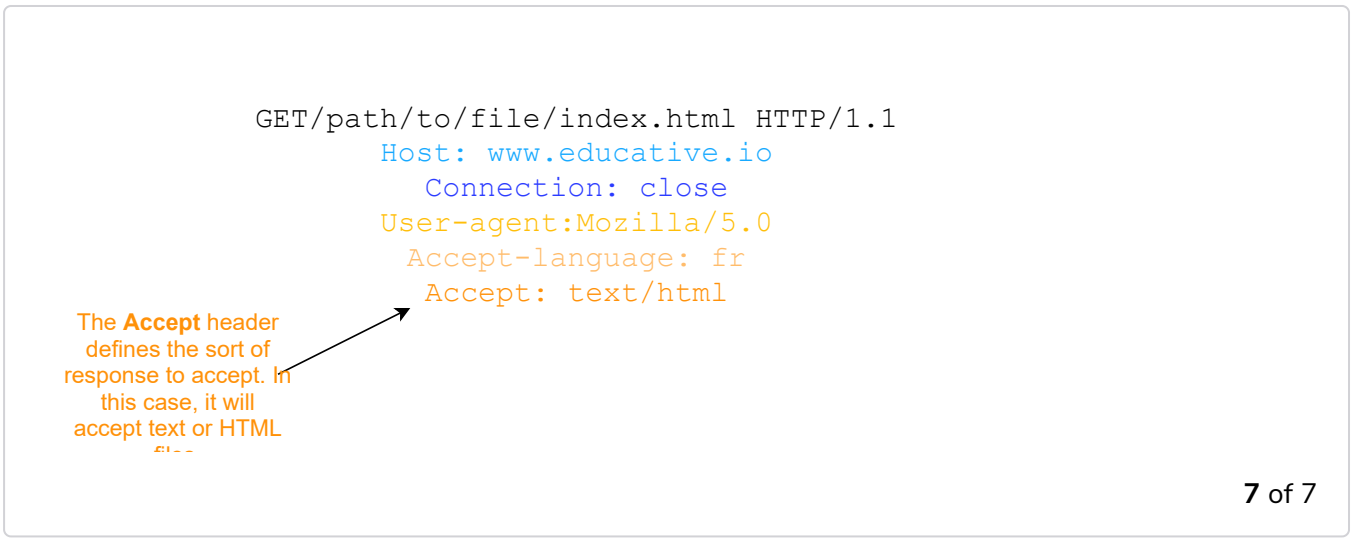
```
GET/path/to/file/index.html HTTP/1.1
Host: www.educative.io
Connection: close
User-agent:Mozilla/5.0
Accept-language: fr
Accept: text/html
```


Request line
GET/path/to/file/index.html HTTP/1.1
Host: www.educative.io
Connection: close
User-agent:Mozilla/5.0
Accept-language: fr
Accept: text/html


host specifies the domain that the request is for, This one was for educative.io
GET/path/to/file/index.html HTTP/1.1
Host: www.educative.io
Connection: close
User-agent:Mozilla/5.0
Accept-language: fr
Accept: text/html


GET/path/to/file/index.html HTTP/1.1
Host: www.educative.io
Connection: close
User-agent:Mozilla/5.0
Accept-language: fr
Accept: text/html
Connection indicates if the connection is to be persistent or not. In this case, it is not persistent and will be closed after every message.





In the next lesson, we'll conduct an exercise to look at real HTTP request messages!