Hypertext Transfer Protocol (HTTP) Fundamentals

Péter Jeszenszky

September 27, 2025

What is HTTP?

- A family of stateless, application-level, request/response protocols that share a generic interface, extensible semantics, and self-descriptive messages to enable flexible interaction with network-based hypertext information systems.
- Initially, it was developed as a joint effort between the IETF and the W3C.
- Currently, it is developed by the IETF HTTP Working Group: https://httpwg.org/

Characteristics

- **Uniform interface**: provides a uniform interface for interacting with resources by sending messages that manipulate or transfer representations.
- **Request/response protocol based on the client-server model**: operates by exchanging messages between clients and servers.
- **Stateless protocol**: each request message's semantics can be understood in isolation.
- **Extensible**: HTTP defines several generic extension points that can be used to introduce capabilities to the protocol without introducing a new version, including methods, status codes, and fields.

Current HTTP Versions

- HTTP/1.1
- HTTP/2
- HTTP/3

Current Standards

- Roy T. Fielding (ed.), Mark Nottingham (ed.), Julian Reschke (ed.). HTTP Semantics. RFC 9110, June 2022.
- Roy T. Fielding (ed.), Mark Nottingham (ed.), Julian Reschke (ed.). HTTP Caching. RFC 9111, June 2022.
- Roy T. Fielding (ed.), Mark Nottingham (ed.), Julian Reschke (ed.). HTTP/1.1. RFC 9112, June 2022.
- Martin Thomson (ed.), Cory Benfield (ed.). HTTP/2. RFC 9113, June 2022.
- Mike Bishop (ed.). HTTP/3. RFC 9114, June 2022.

History (1)

HTTP 0.9:

- The first documented version was written by Tim Berners-Lee in 1991.
- Very simple, supports only GET requests for which an HTML document consisting of ASCII characters is sent back as a response.
- See: https://www.w3.org/Protocols/HTTP/AsImplemented.html

History (2)

HTTP/1.0:

- Tim Berners-Lee, Roy T. Fielding, Henrik Frystyk Nielsen. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945, May 1996.
  - Uses messages that also contain meta-information about enclosed content.
  - Supports not only the transmission of HTML documents but also of any other media types.
  - Supports multiple methods (GET, HEAD, POST, PUT, DELETE, LINK, ULINK).
  - Supports authentication (basic authentication).

History (3)

HTTP/1.1:

- Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2068, January 1997.
  - New features: persistent connections, content negotiation, range requests, ...
- Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, Tim Berners Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999.
  - An update to RFC 2068.

History (4)

HTTP/1.1 (continued):

- RFC 7230, RFC 7231, RFC 7232, RFC 7233, RFC 7234, RFC 7235
  - A new revision of HTTP/1.1 published in 2014 in six RFCs.

History (5)

HTTP/2:

- Described by RFC 7540 and RFC 7541 published in May 2015.
- Differs from HTTP/1.1 in the framing of messages: a binary protocol instead of being a text-based one.
- Introduces many new features, e.g., multiplexing, server push, ...

HTTP/3:

- Described by RFC 9114 and RFC 9204 published in June 2022.
- Very similar to HTTP/2 in respect of the features offered, however, it is not based on TCP anymore. Instead, it is based on QUIC, a transport protocol built on top of UDP.

History (6)

HTTP Semantics:

- In June 2022, the HTTP/1.1, HTTP/2, and HTTP/3 specifications were reformulated based on the *HTTP Semantics* specification, which establishes a common ground for all versions of HTTP.

History (7)

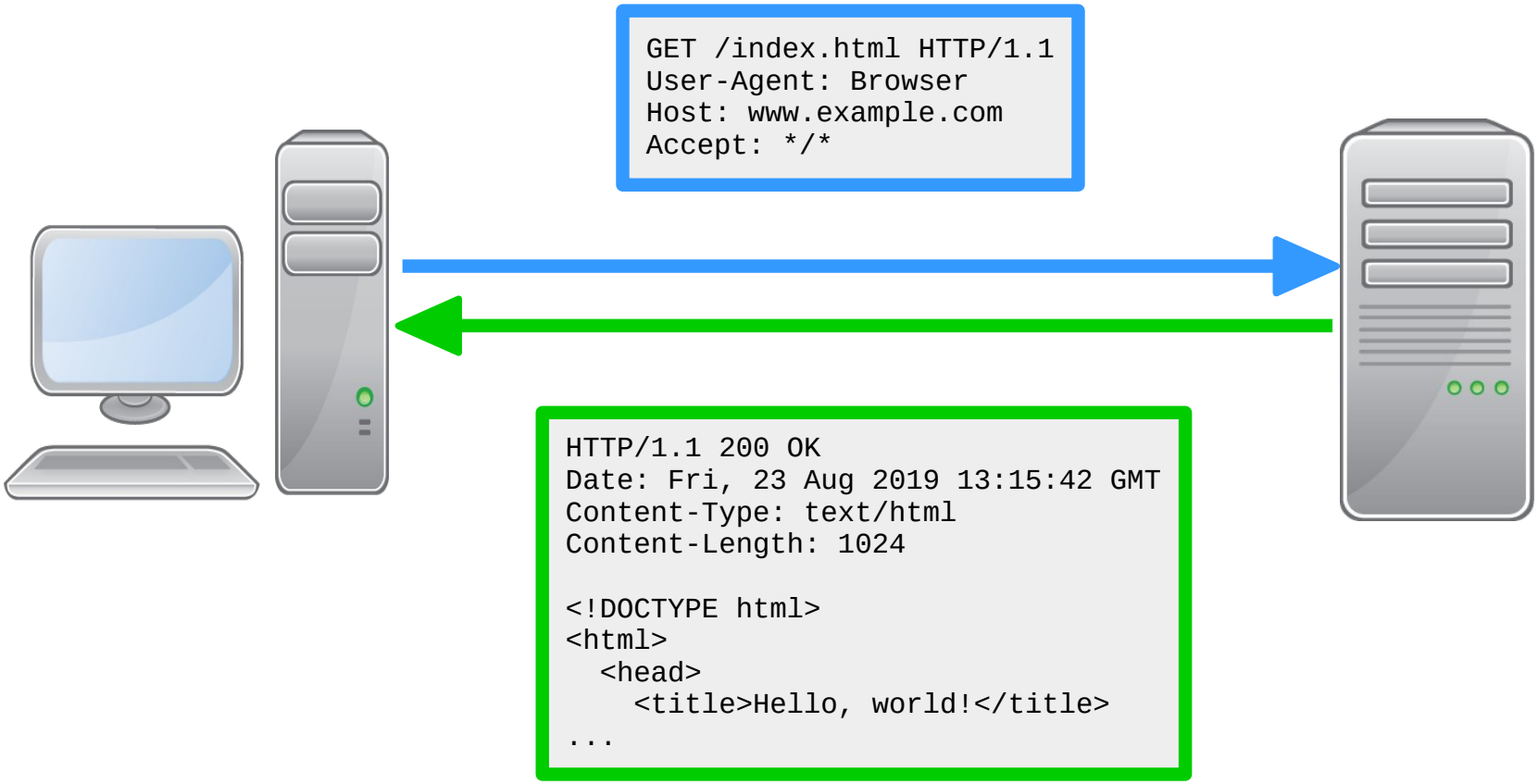
Further reading:

- Evolution of HTTP (MDN)
- High Performance Browser Networking – Brief History of HTTP

HTTP Semantics

- The HTTP Semantics specification (i.e., RFC 9110)
  - describes the overall architecture of HTTP,
  - establishes common terminology,
  - defines aspects of the protocol that are shared by all versions.
- HTTP's core semantics don't change between protocol versions; their expression "on the wire" can change.

How It Works


GET /index.html HTTP/1.1
User-Agent: Browser
Host: www.example.com
Accept: */*
HTTP/1.1 200 OK
Date: Fri, 23 Aug 2019 13:15:42 GMT
Content-Type: text/html
Content-Length: 1024
<!DOCTYPE html>
<html>
<head>
<title>Hello, world!</title>
...

Extending HTTP

Generic extension points:

- Methods
- Status codes
- Fields
- Authentication schemes
- Range units
- Content codings

Secure HTTP

- Connections are secured by TLS (formerly SSL).
- See:
  - Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

curl

Command-line tool (`curl`) and library (`libcurl`) for transferring data specified with URL syntax.

- Website: https://curl.se/
- Repository: https://github.com/curl/curl
- Written in: C
- Platform: Linux, macOS, Windows, ...
- License: X11 License

Supported protocols: FTP(S), HTTP(S), POP3(S), SCP, SMTP(S), ...

HTTPie (1)

Command-line HTTP client.

- Website: https://httpie.io/
- Repository: https://github.com/httpie/cli
- Written in: Python
- Platform: Linux, macOS, Windows
- License: New BSD License

Limitation: supports only HTTP/1.1.

HTTPie (2)

Currently, in beta:

- HTTPie Desktop (platform: Linux, macOS, Windows): https://httpie.io/desktop
- HTTPie for Web: https://httpie.io/app

Browser Tools

- **Chromium**, **Google Chrome**, **Opera**: Chrome DevTools
- **Firefox**: Firefox Developer Tools
- **Chromium-based Edge**: Microsoft Edge DevTools documentation
- **Safari**: https://developer.apple.com/safari/tools/

REST Client (Visual Studio Code)

Visual Studio Code extension that allows users to send HTTP requests and view the corresponding responses in the editor.

- Web page: https://marketplace.visualstudio.com/items?itemName=humao.rest-client
- Repository: https://github.com/Huachao/vscode-restclient
- Written in: TypeScript
- Platform: Visual Studio Code
- License: MIT License

Terminology (1)

- **Resource**: The target of an HTTP request that is identified by a URI.
- **Representation**:
  - Information that is intended to reflect a past, current, or desired state of a given resource.
  - Can be readily communicated via the protocol.
  - Consists of a set of representation metadata and a potentially unbounded stream of representation data.
  - A resource might be provided with, or be capable of generating, multiple representations that are each intended to reflect the resource's current state.

Terminology (2)

- **Content negotiation**: a mechanism for selecting and serving the most appropriate resource representation to satisfy a specific request that is applied when resources can have several alternative representations.

Terminology (3)

- **Connection**: HTTP is a client/server protocol that operates over a reliable transport- or session-layer connection.
- **Message**: HTTP is a stateless request/response protocol for exchanging messages across a connection.
  - In general, a message is either a request or a response. (However, for example, HTTP/2 also uses connection control messages.)
- **Sender/recipient**: any implementation that sends or receives a given message, respectively.

Terminology (4)

The terms client and server refer only to the roles that these programs perform for a particular connection. The same program might act as a client on some connections and a server on others.

- **Client**: a program that establishes a connection to a server for the purpose of sending one or more HTTP requests.
- **Server**: a program that accepts connections to service HTTP requests by sending HTTP responses.

Terminology (5)

- **User agent**: a client program that initiates an HTTP request.
  - E.g., a web browser, web crawler, command-line tool (e.g., `curl`, HTTPie), household appliance, firmware update script, mobile app, ...
  - Being a user agent does not imply that there is a human user directly interacting with the software agent at the time of a request.
- **Origin server**: a program that can originate authoritative responses for a given target resource.

Terminology (6)

- **Intermediary**: allows requests to be satisfied through a chain of connections.
  - There are three common forms of intermediaries: proxy, gateway, tunnel.

Terminology (7)

Example:

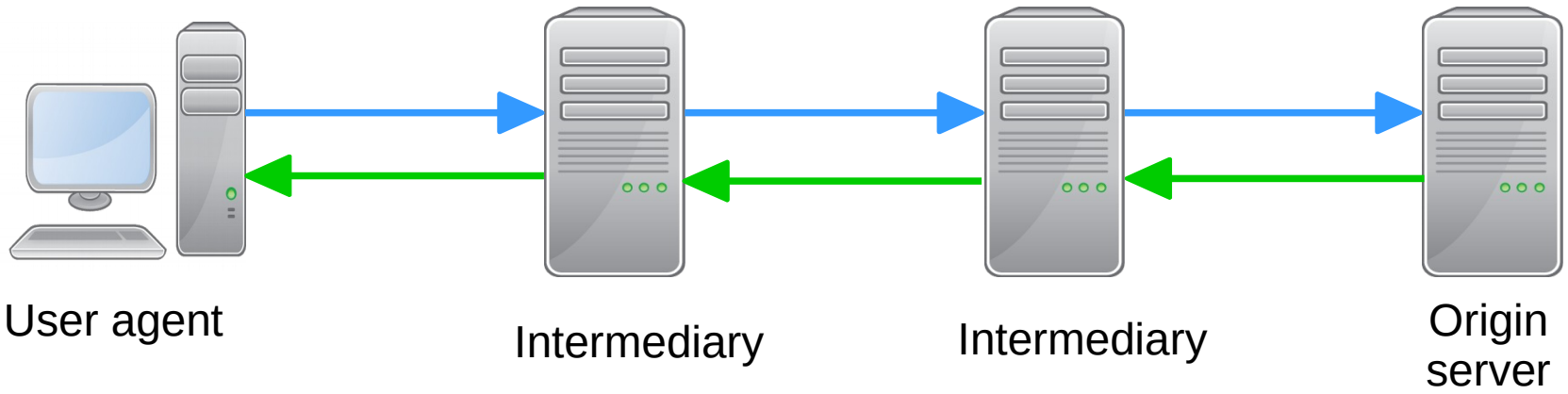


Figure 1: A request or response message that travels the whole chain will pass through three separate connections.

`http` and `https` URI Schemes

- HTTP defines the `http` and `https` URI schemes.
  - The origin server for an `http` or `https` URI is identified by the host identifier and the optional port number.
  - The path component and the optional query component identify a potential target resource within that origin server's namespace.
- The presence of an `http` or `https` URI does not imply that there is always an HTTP server at the identified origin server listening for connections.
- Resources made available via the `https` scheme have no shared identity with the `http` scheme.

`http` URI Scheme

- The `http` URI scheme is defined for the purpose of identifying resources on a potential origin server listening for TCP connections on a given port.
- URI syntax:

  ```
  "http://" host [":" port] [path] ["?" query]
  ```

  - If the port subcomponent is not given or empty, TCP port 80 is the default.

`https` URI Scheme

- The `https` URI scheme is defined for the purpose of identifying resources on a potential origin server listening for TCP connections on a given port and capable of establishing a TLS connection that has been secured for HTTP communication.
- URI syntax:

  ```
  "https://" host [":" port] [path] ["?" query]
  ```

  - If the port subcomponent is not given or empty, TCP port 443 is the default.

`http` and `https` URI comparison (1)

- An empty path component is equivalent to a path of `"/"`.
- The scheme and host are case-insensitive and normally provided in lowercase. All other components are compared in a case-sensitive manner.
- Characters other than those in the “reserved” set are equivalent to their percent-encoded octets.

`http` and `https` URI comparison (2)

- For example, the following URIs are equivalent:
  - http://www.inf.unideb.hu/, http://www.inf.unideb.hu:80/, http://www.inf.unideb.hu, http://www.inf.unideb.hu:80
  - https://web.unideb.hu/~jeszy/, https://web.unideb.hu/%7ejeszy/, https://WEB.UNIDEB.HU/%7Ejeszy/
- HTTP URIs that are equivalent can be assumed to identify the same resource.

Message Framing

- Each major version of HTTP defines its own syntax for communicating messages, also called a **framing mechanism**.

Message Abstraction

- RFC 9110 provides an abstraction over messages, according to which a message consists of the following:
  - Control data
  - Header section
  - Content
  - Trailer section
- This message abstraction is a generalization across many versions of HTTP, including features that might not be found in some versions.
- Messages are intended to be self-descriptive, i.e., everything a recipient needs to know about the message can be determined by looking at the (decoded) message itself.

Message Abstraction: Control Data

- Messages start with control data that describe their primary purpose.
  - Request message control data includes a request method, request target, and protocol version.
  - Response message control data includes a status code, optional reason phrase, and protocol version.
- Every HTTP message has a protocol version.

Message Abstraction: Header Section

- Fields that are sent or received before the content are referred to as **header fields** (or just **headers**, colloquially).
- The header section of a message consists of a sequence of header field lines.

Message Abstraction: Content (1)

- HTTP messages can carry a complete or partial representation as the message content.
- Content is transferred as a stream of octets after the header section.
- It is in a format and encoding defined by the header fields `Content-Type` and `Content-Encoding`.

Message Abstraction: Content (2)

Content semantics:

- The purpose of content in a request is defined by the method semantics.
  - For example, a representation in the content of a POST request represents information to be processed by the target resource.
- In a response, the content's purpose is defined by the request method, response status code, and response fields describing that content.
  - For example, the content of a 200 (OK) response to GET represents the current state of the target resource, as observed at the time of the message origination date.

Message Abstraction: Trailer Section

- Fields that are sent or received after the content are referred to as **trailer fields** (or just **trailers**, colloquially).
- Trailer fields can be used to carry checksums, digital signatures, delivery metrics, or post-processing status information.
- The trailer section of a message consists of a sequence of trailer field lines.
- Trailer fields should be processed and stored separately from header fields.

Fields (1)

- HTTP uses fields to provide data in the form of name/value pairs.
- Fields are used to carry:
  - message metadata both in requests and responses (e.g., `Date`),
  - representation metadata both in requests and responses (e.g., `Content-Type`),
  - client information in requests (e.g., `User-Agent`),
  - server information in responses (e.g., `Server`),
  - resource metadata in responses (e.g., `Last-Modified`).

Fields (2)

- Fields are sent and received within the header and trailer sections of messages.
  - A field sent in the header or trailer section of a message is called a header or trailer field, respectively.
  - Certain fields (such as, e.g., `ETag`) can occur either as a header or a trailer field.

Fields (3)

Field names:

- A field name is a sequence of one or more characters from a subset of the US-ASCII character set.
- Field names are case-insensitive.

Fields (4)

Field values:

- A field value is a sequence of one or more visible US-ASCII characters, spaces, and horizontal tabs.
- Leading and trailing whitespaces must be stripped before consuming.
- Fields can be defined to carry either a single member or a comma-separated list of members.
- Each field can constrain the set of allowed values.

Fields (5)

Field sections:

- Field sections are composed of any number of field lines, each with a field name identifying the field, and a field line value that conveys data for that instance of the field.
- When a field name is repeated within a section, its value consists of the list of corresponding field line values within that section, concatenated in order, with each field line value separated by a comma.
    - In practice, the `Set-Cookie` header field often appears in a response message across multiple field lines; however, `Set-Cookie` field line values of are not combined into a single field value.
- The order in which field lines with differing field names occur in a section is not significant.

Fields (6)

- HTTP specifications define many standard fields.
- Many other specifications define fields beyond the ones specified by HTTP.
- Unless specified otherwise, fields are defined for all versions of HTTP.
- IANA maintains the registry of HTTP fields.
  - See: Hypertext Transfer Protocol (HTTP) Field Name Registry (IANA)

Representation Metadata (1)

- Representation header fields provide metadata about the representation.
- When a message includes content, the representation header fields describe how to interpret that data.
- In a response to a HEAD request, the representation header fields describe the representation data that would have been enclosed in the content if the same request had been a GET.

Representation Metadata (2)

- The `Content-Type` header field indicates the media type of the associated representation.
- Examples:
  - `Content-Type: text/plain`
  - `Content-Type: image/png`
  - `Content-Type: text/html; charset=UTF-8`

Representation Metadata (3)

- The `Content-Encoding` header field indicates what content codings have been applied to the representation, beyond those inherent in the media type.
- Thus, it indicates what decoding mechanisms have to be applied to obtain data in the media type referenced by the `Content-Type` header field.
- Examples:
  - `Content-Encoding: gzip`
  - `Content-Encoding: br`

Content Codings (1)

- Content codings are used to allow a representation to be compressed or otherwise usefully transformed without losing the identity of its underlying media type and without loss of information.
- Frequently, the representation is stored in coded form, transmitted directly, and only decoded by the final recipient.
- The content codings that have been applied are a characteristic of the representation.
  - All other metadata about the representation is about the coded form.

Content Codings (2)

- IANA maintains the registry of content codings:
  - Hypertext Transfer Protocol (HTTP) Parameters – HTTP Content Coding Registry (IANA)
- Content coding names are case-insensitive.

Content Codings (3)

Examples:

- **gzip**: GZIP file format (see RFC 1952)
- **br**: Brotli compressed data format (see RFC 7932)

The `User-Agent` Header Field (1)

- Contains information about the user agent originating the request.
- Can be used for tailoring responses or analytics regarding browser or operating system use.
- A user agent should send a `User-Agent` header field in each request.
- The field value consists of one or more product identifiers, each followed by zero or more comments.
  - Product identifiers are listed in decreasing order of their significance.
  - Each product identifier consists of a name and an optional version.
  - Comments are surrounded by parentheses.

The User-Agent Header Field (2)

Examples:

- **curl**: `curl/8.14.1`
- **Firefox**: `Mozilla/5.0 (X11; Linux x86_64; rv:143.0) Gecko/20100101 Firefox/143.0`
- **Google Chrome**: `Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36`
- **Safari**: `Mozilla/5.0 (Macintosh; Intel Mac OS X 15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/18.6 Safari/605.1.15`

Methods (1)

- Primarily, methods indicate the purpose of requests.
- Method semantics might be further specialized by some header fields in the request (see, e.g., the conditional request header fields).
- Method names are case-sensitive.

Methods (2)

Standardized methods defined by the specification:

- CONNECT
- DELETE
- GET
- HEAD
- OPTIONS
- POST
- PUT
- TRACE

Methods (3)

- All general-purpose servers must support the methods GET and HEAD; all other methods are optional.
- Additional methods have been specified for use in HTTP.
- IANA maintains the registry of HTTP methods.
  - See: Hypertext Transfer Protocol (HTTP) Method Registry (IANA)

Methods (4)

- The set of methods allowed by a target resource can be listed in an `Allow` header field.
- An origin server that receives a request method that is unrecognized or not implemented should respond with the 501 (Not Implemented) status code.
- An origin server that receives a request method that is recognized and implemented, but not allowed for the target resource, should respond with the 405 (Method Not Allowed) status code.

GET method

- Requests transfer of a current selected representation for the target resource.
- The primary mechanism of information retrieval.
- A client can alter the semantics of GET to be a range request, requesting transfer of only some part(s) of the selected representation, by sending a `Range` header field in the request.

HEAD method (1)

- Identical to the GET method except that the server must not send content in the response.
- Can be used to obtain metadata about the selected representation without transferring its representation data.

HEAD method (2)

Example:

```
$ curl --head https://sqlite.org/
HTTP/1.1 200 OK
Connection: keep-alive
Date: Wed, 24 Sep 2025 08:23:23 GMT
Last-Modified: Fri, 19 Sep 2025 22:03:35 GMT
Cache-Control: max-age=120
ETag: "m68cdd337s2467"
Content-type: text/html; charset=utf-8
Content-length: 9319
```

POST method (1)

- Requests that the target resource process the representation enclosed in the request according to the resource's own specific semantics.
- Common uses include:
  - Submitting data (e.g., form data) to a data-handling process.
  - Posting a message to a newsgroup, mailing list, or blog.
  - Creating a new resource.
  - Appending data to a resource's existing representation(s).

POST method (2)

Example: httpbin.org

```
$ http --form https://httpbin.org/post number=42 text="Hello, World!" -v
POST /post HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate, br, zstd
Connection: keep-alive
Content-Length: 32
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Host: httpbin.org
User-Agent: HTTPie/3.2.4

number=42&text=Hello%2C+World%21
```

POST method (3)

Example: httpbin.org

```
$ http --form --multipart https://httpbin.org/post \
  number=42 text="Hello, World!" -v
POST /post HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate, br, zstd
Connection: keep-alive
Content-Length: 225
Content-Type: multipart/form-data; boundary=f067108d371949f2a8c4fdbb991afac1
Host: httpbin.org
User-Agent: HTTPie/3.2.4

--90f9bc6b7be04d918e70eda8ef004ec8
Content-Disposition: form-data; name="number"

42
--90f9bc6b7be04d918e70eda8ef004ec8
Content-Disposition: form-data; name="text"

Hello, World!
--90f9bc6b7be04d918e70eda8ef004ec8--
```

POST method (4)

Examle: Fun Translations API

```
$ http --form https://api.funtranslations.com/translate/yoda.json \
  text="Hello, World!" -v
POST /translate/yoda.json HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate, br, zstd
Connection: keep-alive
Content-Length: 22
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Host: api.funtranslations.com
User-Agent: HTTPie/3.2.4

text=Hello%2C+World%21
```

POST method (5)

Example (continued): Fun Translations API

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Cache-Control: max-age={expires}, pre-check=86400, post-check=43200
Connection: keep-alive
Content-Language: en
Content-Length: 183
Content-Type: application/json; charset=utf-8
Date: Wed, 24 Sep 2025 08:29:52 GMT
Expires: 0
Server: nginx/1.14.0 (Ubuntu)

{
    "contents": {
        "text": "Hello, World!",
        "translated": "Force be with you,World!",
        "translation": "yoda"
    },
    "success": {
        "total": 1
    }
}
```

PUT method (1)

- Requests that the state of the target resource be created or replaced with the state defined by the representation enclosed in the request message content.
- A successful PUT of a given representation would suggest that a subsequent GET on that same target resource will result in an equivalent representation being sent in a 200 (OK) response.

PUT method (2)

Example: transfer.sh

```
$ echo "Hello, World!" | http PUT https://transfer.whalebone.io/hello.txt \
  Content-Type:text/plain -v
PUT /hello.txt HTTP/1.1
Accept: application/json, */*;q=0.5
Accept-Encoding: gzip, deflate, br, zstd
Connection: keep-alive
Content-Length: 14
Content-Type: text/plain
Host: transfer.whalebone.io
User-Agent: HTTPie/3.2.4

Hello, World!
```

PUT method (3)

Example (continued): transfer.sh

```
HTTP/1.1 200 OK
Content-Length: 50
Content-Type: text/plain
Date: Sat, 27 Sep 2025 11:10:59 GMT
Server: Transfer.sh HTTP Server
X-Made-With: <3 by DutchCoders
X-Served-By: Proudly served by DutchCoders
X-Url-Delete: https://transfer.whalebone.io/ve7w0ws5oG/hello.txt/6JfDfr1EWsizYS44Vi

https://transfer.whalebone.io/ve7w0ws5oG/hello.txt
```

Difference between PUT and POST

The fundamental difference between the POST and PUT methods is highlighted by the different intent for the enclosed representation:

- The target resource in a POST request is intended to handle the enclosed representation according to the resource's own semantics.
- The enclosed representation in a PUT request is defined as replacing the state of the target resource.

DELETE method (1)

- Requests that the origin server remove the association between the target resource and its current functionality.
- If the target resource has one or more current representations, they might or might not be destroyed by the origin server, and the associated storage might or might not be reclaimed, depending entirely on the nature of the resource and its implementation by the origin server.
- Relatively few resources allow the DELETE method.

DELETE method (2)

Example: transfer.sh

```
$ http https://transfer.whalebone.io/ve7w0ws5oG/hello.txt -v
DELETE /ve7w0ws5oG/hello.txt HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate, br, zstd
Connection: keep-alive
Content-Length: 0
Host: transfer.whalebone.io
User-Agent: HTTPie/3.2.4

HTTP/1.1 405 Method Not Allowed
Content-Length: 0
Date: Sat, 27 Sep 2025 11:15:14 GMT
Server: Transfer.sh HTTP Server
X-Made-With: <3 by DutchCoders
X-Served-By: Proudly served by DutchCoders
```

OPTIONS method (1)

- Requests information about the communication options available for the target resource, at either the origin server or an intervening intermediary.
- * as the request-target applies to the server in general rather than to a specific resource.
- In a successful response to an OPTIONS request, a server should send any header fields that might indicate optional features implemented by the server and applicable to the target resource.
  - For example, the `Allow` header field lists the set of methods advertised as supported by the target resource.

OPTIONS method (2)

Example:

```
$ curl -X OPTIONS https://httpbin.org/anything -v
> OPTIONS /anything HTTP/1.1
> Host: httpbin.org
> User-Agent: curl/8.14.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sat, 27 Sep 2025 07:18:51 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 0
< Connection: keep-alive
< Server: gunicorn/19.9.0
< Allow: POST, TRACE, PUT, PATCH, DELETE, HEAD, OPTIONS, GET
< ...
```

TRACE method

- Requests the request message to be sent back.
  - The final recipient of the request should reflect the message received back to the client as the content of a 200 (OK) response. The `message/http` format is one way to do so.
  - Request fields that are likely to contain sensitive data should be excluded.
- In general, the method is not allowed on servers for security reasons.

Status Codes (1)

- The status code of a response is a three-digit integer code that describes the result of the request and the semantics of the response, including whether the request was successful and what content is enclosed (if any).
- All valid status codes are within the range of 100 to 599, inclusive.

Status Codes (2)

The first digit of the status code defines the class of response:

- 1*xx* (Informational): indicates an interim response for communicating connection status or request progress before sending a final response.
- 2*xx* (Successful): indicates that the request was successfully received, understood, and accepted.
- 3*xx* (Redirection): indicates that further action needs to be taken by the user agent to fulfill the request that can be performed automatically by the user agent.
- 4*xx* (Client Error): indicates that the request contains bad syntax or cannot be fulfilled.
- 5*xx* (Server Error): indicates that the server failed to fulfill an apparently valid request.

Status Codes (3)

- A client is not required to understand the meaning of all registered status codes.
  - However, a client must understand the class of any status code, as indicated by the first digit.
  - An unrecognized status code must be treated as being equivalent to the *x*00 status code of that class.
- Except when responding to a HEAD request in a response with a status code of 4*xx* or 5*xx*, the server should send a representation containing an explanation of the error situation, and whether it is a temporary or permanent condition.

Status Codes (4)

- Status codes are extensible.
- IANA maintains the registry of status codes.
  - See: Hypertext Transfer Protocol (HTTP) Status Code Registry (IANA)

Major Status Codes (1)

Status Code	Reason	Description
100	Continue	The initial part of a request has been received and has not yet been rejected by the server. The server intends to send a final response after the request has been fully received and acted upon.
101	Switching Protocols	The server is willing to change the application protocol being used on this connection.

Major Status Codes (2)

Status Code	Reason	Description
200	OK	The request has succeeded. The content sent in the response depends on the request method. For example, in a response to a GET request, the content is a representation of the target resource.
201	Created	The request has been fulfilled, resulting in the creation of one or more new resources.
202	Accepted	The request has been accepted for processing, but the processing has not been completed.
204	No Content	The server has successfully fulfilled the request, and there is no additional content to send in the response content.
206	Partial Content	The server is successfully fulfilling a range request for the target resource by transferring one or more parts of the selected representation.

Major Status Codes (3)

Status Code	Reason	Description
300	Multiple Choices	The target resource has multiple representations. For request methods other than HEAD, the server should generate content in a 300 response containing a list of representation metadata and URI reference(s) from which the user or user agent can choose the one most preferred.
301	Moved Permanently	The target resource has been assigned a new permanent URI, and any future references to this resource ought to use one of the enclosed URIs.
302	Found	The target resource resides temporarily under a different URI.
303	See Other	The server is redirecting the user agent to a different resource, which is intended to provide an indirect response to the original request.
304	Not Modified	A conditional GET or HEAD request has been received, and there is no need for the server to transfer a representation of the target resource because the request indicates that the client, which made the request conditional, already has a valid representation.

Major Status Codes (4)

Status Code	Reason	Description
400	Bad Request	The server cannot or will not process the request due to an issue perceived as a client error (e.g., malformed request syntax).
401	Unauthorized	The request lacks valid authentication credentials for the target resource.
403	Forbidden	The server understood the request but refused to fulfill it. If authentication credentials were provided in the request, the server considers them insufficient to grant access.
404	Not Found	The origin server did not find a current representation for the target resource or is not willing to disclose that one exists.
405	Method Not Allowed	The request method is known by the origin server but not supported by the target resource.

Major Status Codes (5)

Status Code	Reason	Description
500	Internal Server Error	The server encountered an unexpected condition that prevented it from fulfilling the request.
501	Not Implemented	The server does not support the functionality (e.g., request method) required to fulfill the request.
503	Service Unavailable	The server is currently unable to handle the request due to, e.g., a temporary overload or scheduled maintenance.

Fun with Status Codes

Let's have some fun:

- 418 (I'm a teapot):
  - Larry Masinter. Hyper Text Coffee Pot Control Protocol (HTCPCP/1.0). RFC 2324, 1 April 1998.
  - Imran Nazar. The Hyper Text Coffee Pot Control Protocol for Tea Efflux Appliances (HTCPCP-TEA). RFC 7168, 1 April 2014.
- HTTP Cats
- HTTP Status Dogs

Redirection (1)

Example:

```
$ curl --http1.1 -v http://w3.org/
> GET / HTTP/1.1
> Host: w3.org
> User-Agent: curl/8.14.1
> Accept: */*
>
< HTTP/1.1 301 Moved Permanently
< Location: https://www.w3.org/
< ...
<
```

Redirection (2)

Example:

```
$ curl --http1.1 -v -L http://w3.org/
> Host: w3.org
> User-Agent: curl/8.14.1
> Accept: */*
>
< HTTP/1.1 301 Moved Permanently
< Location: https://www.w3.org/
< ...
<
> GET / HTTP/1.1
> Host: www.w3.org
> User-Agent: curl/8.14.1
> Accept: */*
>
< HTTP/1.1 200 OK
< ...
```

Redirection (3)

Example:

```
$ http https://dbpedia.org/resource/Hungary
HTTP/1.1 303 See Other
Connection: keep-alive
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Date: Sat, 27 Sep 2025 06:59:02 GMT
Location: http://dbpedia.org/page/Hungary
Server: Virtuoso/08.03.3332 (Linux) x86_64-generic-linux-glibc212  VDB
...
```

Content Negotiation

- When responses convey content, whether indicating a success or an error, the origin server often has different ways of representing that information; for example, in various formats, languages, or encodings.
- Likewise, different users or user agents might have differing capabilities, characteristics, or preferences that could influence which representation, among those available, would be best to deliver.
- For this reason, HTTP provides mechanisms for content negotiation.

Patterns of Content Negotiation

HTTP defines the following two major patterns of content negotiation that can be made visible within the protocol:

- **Proactive negotiation**: the server selects the representation based upon the user agent's stated preferences.
  - It is also known as server-driven negotiation.
- **Reactive negotiation**: the server provides a list of representations for the user agent to choose from.
  - It is also known as agent-driven negotiation.

Other patterns of content negotiation are possible. These patterns are not mutually exclusive.

Proactive Negotiation (1)

- The origin server uses an algorithm to select the preferred representation based on the preferences of the user agent.
- Selection is based on the available representations for a response compared to various information supplied in the request, including both the explicit negotiation header fields and implicit characteristics, such as the client's network address or parts of the `User-Agent` field.
- A `Vary` header field is often sent in a response subject to proactive negotiation to indicate what parts of the request information were used in the selection algorithm.

Proactive Negotiation (2)

Advantageous:

- When the algorithm for selecting from among the available representations is difficult to describe to a user agent, or
- When the server desires to send its “best guess” to the user agent along with the first response, to avoid a subsequent request.

Proactive Negotiation (3)

Disadvantages:

- It is impossible for the server to accurately determine what might be "best" for any given user, since that would require complete knowledge of both the capabilities of the user agent and the intended use for the response.
- Having the user agent describe its capabilities in every request can be both very inefficient and a potential risk to the user's privacy.
- Complicates the implementation of an origin server and the algorithms for generating responses to a request.
- Limits the reusability of responses for shared caching.

Reactive Negotiation (1)

- The user agent performs selection of content after receiving an initial response that contains a list of resources for alternative representations.
- Selection of alternatives might be performed automatically by the user agent or manually by the user.

Reactive Negotiation (2)

- Advantageous:
  - When the response would vary over commonly used dimensions (such as type, language, or encoding), or
  - when the origin server is unable to determine a user agent's capabilities from examining the request.
- Disadvantages:
  - After obtaining the list of alternative representations, the user agent must make a second request to get the desired representation.
  - HTTP does not define a mechanism for supporting automatic selection.

q parameter

- Content negotiation fields use a common parameter named q to assign a relative "weight" to the preference for that associated kind of content.
- The weight is normalized to a real number in the range 0 through 1, where 0.001 is the least preferred and 1 is the most preferred; a value of 0 means "not acceptable".
- If no q parameter is present, the default weight is 1.

Content Negotiation Fields

User agents can use the following header fields in requests to specify their preferences:

- **`Accept`**: can be used to specify preferences regarding response media types.
- **`Accept-Charset`**: can be used to indicate preferences for charsets in textual response content.
- **`Accept-Encoding`**: can be used to indicate preferences regarding the use of content codings.
- **`Accept-Language`**: can be used to indicate the set of natural languages that are preferred in the response.

A request that does not contain any of these fields implies that the sender has no preference on that dimension of negotiation.

`Accept` Header Field (1)

- Its value is a comma-separated list of media ranges, where each media range might be followed by zero or more media type parameters (e.g., `charset`), and an optional relative weight (i.e., `q` parameter).
- Media range:
  - `*/*`: indicates all media types
  - *type*`/*`: indicates all subtypes of that type (e.g., `text/*`)
  - *type*/*subtype*: indicates a specific media type (e.g., `text/html`)

`Accept` Header Field (2)

- The value of the field can vary, e.g., it can be different when fetching a document entered in the address bar or an image linked via an `<img>` HTML element.
  - See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Content_negotiation#the_accept_header
- Default values for specific browsers: List of default Accept values (MDN)

Accept Header Field (3)

Default value for Firefox 92 and later:

```
Accept: text/html,application/xhtml+xml,
application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
```

Media Range	q
text/html	1
application/xhtml+xml	1
image/avif	1
image/webp	1
application/xml	0.9
/	0.8

Content Negotiation (1)

Example:

```
$ curl http://www.gnu.org/ -H Accept-Language:de -v -o gnu.de.html
> GET / HTTP/1.1
> Host: www.gnu.org
> User-Agent: curl/8.14.1
> Accept: */*
> Accept-Language:de
>
< HTTP/1.1 200 OK
< Date: Sat, 27 Sep 2025 06:51:38 GMT
< Server: Apache
< Content-Location: home.de.html
< Vary: negotiate,accept-language,Accept-Encoding
< Accept-Ranges: bytes
< Cache-Control: max-age=0
< Expires: Sat, 27 Sep 2025 06:51:38 GMT
< Content-Length: 32321
< Content-Type: text/html
< Content-Language: de
< ...
```

Content Negotiation (2)

Example:

```
$ http https://www.mozilla.org/ Accept-Language:es -v
GET / HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: es
Connection: keep-alive
Host: www.mozilla.org
User-Agent: HTTPie/3.2.4

HTTP/1.1 302 Found
Accept-Ranges: bytes
Age: 65
Connection: keep-alive
Content-Length: 0
Date: Sat, 27 Sep 2025 06:56:08 GMT
Vary: Accept-Language
content-type: text/html; charset=utf-8
location: /es-ES/
server: granian
...
```

Content Negotiation (3)

Example: DBpedia

- Requesting data in JSON format:

```
$ curl http://dbpedia.org/resource/Hungary \
  -H Accept:application/json -L -o Hungary.json
```

- Requesting data in XML format:

```
$ curl https://dbpedia.org/resource/Hungary \
  -H Accept:application/rdf+xml -L -o Hungary.xml
```

- Requesting data in Turtle format:

```
$ curl https://dbpedia.org/resource/Hungary \
  -H Accept:text/turtle -L -o Hungary.ttl
```

HTTP/1.1

- A text-based protocol.
- An HTTP/1.1 message is either a request or a response.

HTTP/1.1: Message Format (1)

- Messages begin with a **start-line** followed by a CRLF.
- The start-line is followed by a sequence of zero or more header field lines, collectively referred to as the **headers** or the **header section**.
- An empty line indicates the end of the header section.
- Optionally, a **message body** may appear at the end of the message.

HTTP/1.1: Message Format (2)

- The start-line of requests has the following syntax:

  `<method> <request-target> "HTTP/1.1"`

  - The request-target identifies the target resource upon which to apply the request.
    - The most common form is the following: `path ["?" query]`.
    - If the target URI's path component is empty, the client must send `"/"` as the path.
    - The host and port components of the target URI are sent in the `Host` header field.

HTTP/1.1: Message Format (3)

- The first line of a response is known as the **status line** and it has the following syntax:

  `"HTTP/1.1" <status-code> [reason-phrase]`

  - The optional reason phrase is a textual phrase describing the status code.
  - A client should ignore the reason phrase because it is not a reliable channel for information.

HTTP/1.1: Message Format (4)

Field lines:

- Each field line consists of a case-insensitive field name followed by a colon (":"), optional leading whitespace, the field line value, and optional trailing whitespace, and is terminated by a CRLF.

HTTP/1.1: Message Format (5)

Message body:

- The message body (if any) of an HTTP/1.1 message is used to carry content for the request or response.
- The message body is identical to the content unless a transfer coding (e.g., `chunked`) has been applied.

Web Server Software (1)

Market share of web servers:

- Netcraft. August 2025 Web Server Survey.

Web Server Software (2)

The most widely used web servers:

Name	Platform	License	Comment
Apache HTTP Server https://httpd.apache.org/	cross-platform	Apache License 2.0	
nginx https://nginx.org/	cross-platform	2-clause BSD / proprietary	Pronunciation: *engine x*
Internet Information Services (IIS) https://www.iis.net/	Windows	proprietary	
Google Web Server (GWS)	Linux	proprietary	Custom developed web server used by Google.

Java Support

- **Pre-JDK 11**: The `java.net.HttpURLConnection` class provides basic HTTP client functionality.
- **JDK 11-**: The `java.net.http` package provides extensive HTTP client functionality.

Client Libraries

- Java:
  - Eclipse Jetty HTTP Client (license: Eclipse Public License 2.0/Apache License 2.0) https://jetty.org/ https://github.com/eclipse/jetty.project
  - OkHttp (license: Apache License 2.0) https://square.github.io/okhttp/ https://github.com/square/okhttp
  - Retrofit (licenc: Apache License 2.0) https://square.github.io/retrofit/ https://github.com/square/retrofit
- Python:
  - Requests (license: Apache License 2.0) https://requests.readthedocs.io/ https://github.com/psf/requests
  - urllib3 (license: MIT License) https://urllib3.readthedocs.io https://github.com/urllib3/urllib3

Further Recommended Reading

- MDN Web Docs – HTTP
- Ilya Grigorik. High Performance Browser Networking. O'Reilly, 2013.