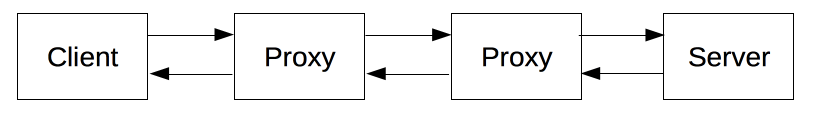
**HTTP**

**Hypertext Transfer Protocol (HTTP)** is an application-layer protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers and web servers, but it can also be used for other purposes. HTTP follows a classical **client-server** model, with a client opening a connection to make a request, then waiting until it receives a response. HTTP is a **stateless protocol**, meaning that the server does not keep any data (state) between two requests.



Clients and servers communicate by exchanging individual messages (as opposed to a stream of data). The messages sent by the client, usually a Web browser, are called requests and the messages sent by the server as an answer are called responses.

Between the client and the server there are numerous entities, collectively called proxies, which perform different operations and act as gateways or caches



[Client: the user-agent](https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#client_the_user-agent)

The *user-agent* is any tool that acts on behalf of the user.

### [The Web server](https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#the_web_server)

On the opposite side of the communication channel is the server, which serves the document as requested by the client. A server appears as only a single machine virtually; but it may actually be a collection of servers sharing the load (load balancing), or a complex piece of software interrogating other computers (like cache, a DB server, or e-commerce servers), totally or partially generating the document on demand.

### [Proxies](https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#proxies)

Those operating at the application layers are generally called **proxies**. These can be transparent, forwarding on the requests they receive without altering them in any way, or non-transparent, in which case they will change the request in some way before passing it along to the server. Proxies may perform numerous functions:

* caching (the cache can be public or private, like the browser cache)
* filtering (like an antivirus scan or parental controls)
* load balancing (to allow multiple servers to serve different requests)
* authentication (to control access to different resources)
* logging (allowing the storage of historical information)

## **Basic Architecture**

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



### 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.

## [**Basic aspects of HTTP**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#basic_aspects_of_http)

### [HTTP is simple](https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#http_is_simple)

HTTP is generally designed to be simple and human readable, even with the added complexity introduced in HTTP/2 by encapsulating HTTP messages into frames. HTTP messages can be read and understood by humans, providing easier testing for developers, and reduced complexity for newcomers.

### [HTTP is extensible](https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#http_is_extensible)

Introduced in HTTP/1.0, HTTP headers make this protocol easy to extend and experiment with. New functionality can even be introduced by a simple agreement between a client and a server about a new header's semantics.

### [HTTP is stateless, but not sessionless](https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#http_is_stateless_but_not_sessionless)

HTTP is stateless: there is no link between two requests being successively carried out on the same connection. This immediately has the prospect of being problematic for users attempting to interact with certain pages coherently, for example, using e-commerce shopping baskets. But while the core of HTTP itself is stateless, HTTP cookies allow the use of stateful sessions. Using header extensibility, HTTP Cookies are added to the workflow, allowing session creation on each HTTP request to share the same context, or the same state.

### [HTTP and connections](https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#http_and_connections)

A connection is controlled at the transport layer, and therefore fundamentally out of scope for HTTP. HTTP doesn't require the underlying transport protocol to be connection-based; it only requires it to be reliable, or not lose messages (at minimum, presenting an error in such cases). Among the two most common transport protocols on the Internet, TCP is reliable and UDP isn't. HTTP therefore relies on the TCP standard, which is connection-based.

Before a client and server can exchange an HTTP request/response pair, they must establish a TCP connection, a process which requires several round-trips. The default behavior of HTTP/1.0 is to open a separate TCP connection for each HTTP request/response pair. This is less efficient than sharing a single TCP connection when multiple requests are sent in close succession.

In order to mitigate this flaw, HTTP/1.1 introduced pipelining (which proved difficult to implement) and persistent connections: the underlying TCP connection can be partially controlled using the Connection header. HTTP/2 went a step further by multiplexing messages over a single connection, helping keep the connection warm and more efficient.

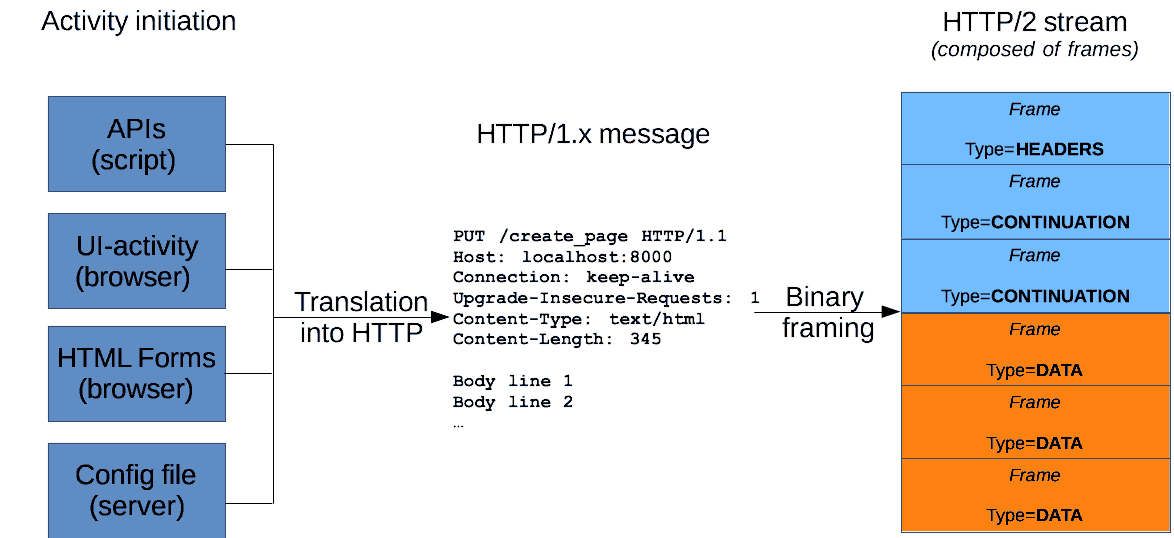
Experiments are in progress to design a better transport protocol more suited to HTTP. For example, Google is experimenting with QUIC which builds on UDP to provide a more reliable and efficient transport protocol.

# **HTTP Messages**

HTTP messages are how data is exchanged between a server and a client. There are two types of messages: requests sent by the client to trigger an action on the server, and responses, the answer from the server.

HTTP messages are composed of textual information encoded in ASCII, and span over multiple lines. In HTTP/1.1, and earlier versions of the protocol, these messages were openly sent across the connection. In HTTP/2, the once human-readable message is now divided up into HTTP frames, providing optimization and performance improvements.

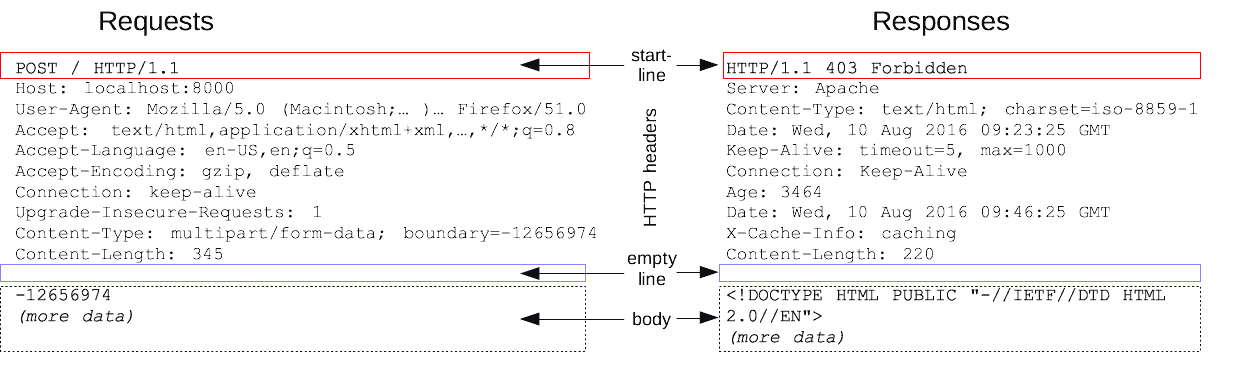
Web developers, or webmasters, rarely craft these textual HTTP messages themselves: software, a Web browser, proxy, or Web server, perform this action. They provide HTTP messages through config files (for proxies or servers), APIs (for browsers), or other interfaces.



HTTP requests, and responses, share similar structure and are composed of:

1. A *start-line* describing the requests to be implemented, or its status of whether successful or a failure. This start-line is always a single line.
2. An optional set of *HTTP headers* specifying the request, or describing the body included in the message.
3. A blank line indicating all meta-information for the request has been sent.
4. An optional *body* containing data associated with the request (like content of an HTML form), or the document associated with a response. The presence of the body and its size is specified by the start-line and HTTP headers.

The start-line and HTTP headers of the HTTP message are collectively known as the *head* of the requests, whereas its payload is known as the *body*.



## [**HTTP Requests**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages#http_requests)

### [Start line](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages#start_line)

HTTP requests are messages sent by the client to initiate an action on the server. Their start-line contain three elements:

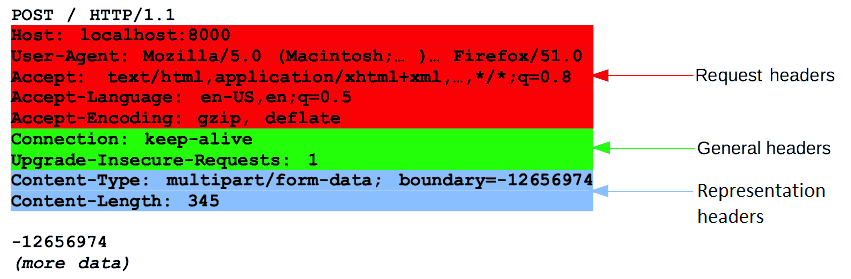
1. An *HTTP method*, a verb (like [GET](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET), [PUT](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT) or [POST](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST)) or a noun (like [HEAD](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/HEAD) or [OPTIONS](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/OPTIONS)), that describes the action to be performed. For example, GET indicates that a resource should be fetched or POST means that data is pushed to the server (creating or modifying a resource, or generating a temporary document to send back).
2. The request target, usually a [URL](https://developer.mozilla.org/en-US/docs/Glossary/URL), or the absolute path of the protocol, port, and domain are usually characterized by the request context. The format of this request target varies between different HTTP methods. It can be
   * An absolute path, ultimately followed by a '?' and query string. This is the most common form, known as the origin form, and is used with GET, POST, HEAD, and OPTIONS methods. POST / HTTP/1.1 GET /background.png HTTP/1.0 HEAD /test.html?query=alibaba HTTP/1.1 OPTIONS /anypage.html HTTP/1.0
   * A complete URL, known as the absolute form, is mostly used with GET when connected to a proxy. GET [https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages HTTP/1.1](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages%20HTTP/1.1)
   * The authority component of a URL, consisting of the domain name and optionally the port (prefixed by a ':'), is called the *authority form*. It is only used with CONNECT when setting up an HTTP tunnel. CONNECT developer.mozilla.org:80 HTTP/1.1
   * The *asterisk form*, a simple asterisk ('\*') is used with OPTIONS, representing the server as a whole. OPTIONS \* HTTP/1.1
3. The *HTTP version*, which defines the structure of the remaining message, acting as an indicator of the expected version to use for the response.

### [Headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages#headers)

[HTTP headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers) from a request follow the same basic structure of an HTTP header: a case-insensitive string followed by a colon (':') and a value whose structure depends upon the header. The whole header, including the value, consist of one single line, which can be quite long.

Many different headers can appear in requests. They can be divided in several groups:

* [General headers](https://developer.mozilla.org/en-US/docs/Glossary/General_header), like [Via](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Via), apply to the message as a whole.
* [Request headers](https://developer.mozilla.org/en-US/docs/Glossary/Request_header), like [User-Agent](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent) or [Accept](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept), modify the request by specifying it further (like [Accept-Language](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Language)), by giving context (like [Referer](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referer)), or by conditionally restricting it (like [If-None](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/If-None)).
* [Representation headers](https://developer.mozilla.org/en-US/docs/Glossary/Representation_header) like [Content-Type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type) that describe the original format of the message data and any encoding applied (only present if the message has a body).



### [Body](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages#body)

The final part of the request is its body. Not all requests have one: requests fetching resources, like GET, HEAD, DELETE, or OPTIONS, usually don't need one. Some requests send data to the server in order to update it: as often the case with POST requests (containing HTML form data).

Bodies can be broadly divided into two categories:

* Single-resource bodies, consisting of one single file, defined by the two headers: [Content-Type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type) and [Content-Length](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Length).
* [Multiple-resource bodies](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types#multipartform-data), consisting of a multipart body, each containing a different bit of information. This is typically associated with [HTML Forms](https://developer.mozilla.org/en-US/docs/Learn/Forms).

## [**HTTP Responses**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages#http_responses)

### [Status line](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages#status_line)

The start line of an HTTP response, called the status line, contains the following information:

1. The protocol version, usually HTTP/1.1.
2. A status code, indicating success or failure of the request. Common status codes are [200](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200), [404](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404), or [302](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/302)
3. A status text. A brief, purely informational, textual description of the status code to help a human understand the HTTP message.

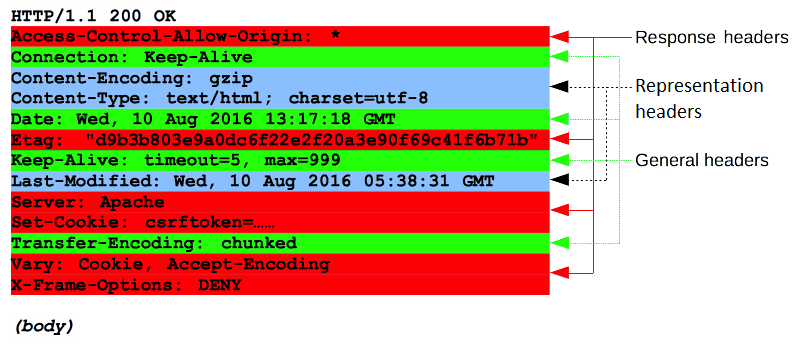
A typical status line looks like: HTTP/1.1 404 Not Found.

[Headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages#headers_2)

### [HTTP headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers) for responses follow the same structure as any other header: a case-insensitive string followed by a colon (':') and a value whose structure depends upon the type of the header. The whole header, including its value, presents as a single line.

Many different headers can appear in responses. These can be divided into several groups:

* [General headers](https://developer.mozilla.org/en-US/docs/Glossary/General_header), like [Via](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Via), apply to the whole message.
* [Response headers](https://developer.mozilla.org/en-US/docs/Glossary/Response_header), like [Vary](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Vary) and [Accept-Ranges](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Ranges), give additional information about the server which doesn't fit in the status line.
* [Representation headers](https://developer.mozilla.org/en-US/docs/Glossary/Representation_header) like [Content-Type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type) that describe the original format of the message data and any encoding applied (only present if the message has a body).



### [Body](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages#body_2)

The last part of a response is the body. Not all responses have one: responses with a status code that sufficiently answers the request without the need for corresponding payload (like [201](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/201) **Created** or [204](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/204) **No Content**) usually don't.

Bodies can be broadly divided into three categories:

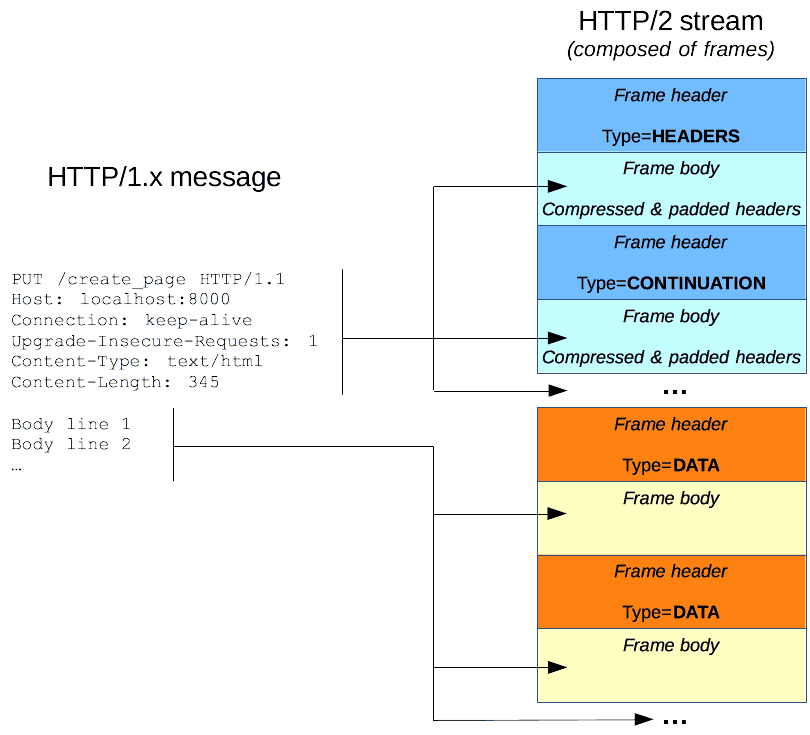
* Single-resource bodies, consisting of a single file of known length, defined by the two headers: [Content-Type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type) and [Content-Length](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Length).
* Single-resource bodies, consisting of a single file of unknown length, encoded by chunks with [Transfer-Encoding](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Transfer-Encoding) set to chunked.
* [Multiple-resource bodies](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types#multipartform-data), consisting of a multipart body, each containing a different section of information. These are relatively rare.

## [HTTP/2 Frames](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages#http2_frames)

HTTP/1.x messages have a few drawbacks for performance:

* Headers, unlike bodies, are uncompressed.
* Headers are often very similar from one message to the next one, yet still repeated across connections.
* No multiplexing can be done. Several connections need opening on the same server: and warm TCP connections are more efficient than cold ones.

HTTP/2 introduces an extra step: it divides HTTP/1.x messages into frames which are embedded in a stream. Data and header frames are separated, which allows header compression. Several streams can be combined together, a process called multiplexing, allowing more efficient use of underlying TCP connections.



HTTP frames are now transparent to Web developers. This is an additional step in HTTP/2, between HTTP/1.1 messages and the underlying transport protocol. No changes are needed in the APIs used by Web developers to utilize HTTP frames; when available in both the browser and the server, HTTP/2 is switched on and used.

**HTTP request methods**

1. [CONNECT](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/CONNECT)
2. [DELETE](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/DELETE)
3. [GET](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET)
4. [HEAD](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/HEAD)
5. [OPTIONS](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/OPTIONS)
6. [PATCH](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH)
7. [POST](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST)
8. [PUT](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT)
9. [TRACE](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/TRACE)

# CONNECT

The **HTTP CONNECT method** starts two-way communications with the requested resource. It can be used to open a tunnel.

For example, the CONNECT method can be used to access websites that use [SSL](https://developer.mozilla.org/en-US/docs/Glossary/SSL) ([HTTPS](https://developer.mozilla.org/en-US/docs/Glossary/https)). The client asks an HTTP [Proxy server](https://developer.mozilla.org/en-US/docs/Glossary/Proxy_server) to tunnel the [TCP](https://developer.mozilla.org/en-US/docs/Glossary/Transmission_Control_Protocol_(TCP)) connection to the desired destination. The server then proceeds to make the connection on behalf of the client. Once the connection has been established by the server, the [Proxy server](https://developer.mozilla.org/en-US/docs/Glossary/Proxy_server) continues to proxy the [TCP](https://developer.mozilla.org/en-US/docs/Glossary/Transmission_Control_Protocol_(TCP)) stream to and from the client.

CONNECT is a hop-by-hop method.

## [Syntax](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/CONNECT#syntax)

CONNECT www.example.com:443 HTTP/1.1

# DELETE

The **HTTP DELETE request method** deletes the specified resource.

## [Syntax](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/DELETE#syntax)

DELETE /file.html HTTP/1.1

### [Responses](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/DELETE#responses)

If a DELETE method is successfully applied, there are several response status codes possible:

* A [202](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/202) (Accepted) status code if the action will likely succeed but has not yet been enacted.
* A [204](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/204) (No Content) status code if the action has been enacted and no further information is to be supplied.
* A [200](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200) (OK) status code if the action has been enacted and the response message includes a representation describing the status.

# GET

The **HTTP GET method** requests a representation of the specified resource. Requests using GET should only be used to request data (they shouldn't include data).

**Note:** Sending body/payload in a GET request may cause some existing implementations to reject the request — while not prohibited by the specification, the semantics are undefined. It is better to just avoid sending payloads in GET requests.

## [Syntax](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET#syntax)

GET /index.html

# HEAD

The **HTTP HEAD method** requests the [headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers) that would be returned if the HEAD request's URL was instead requested with the HTTP [GET](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET) method. For example, if a URL might produce a large download, a HEAD request could read its [Content-Length](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Length) header to check the filesize without actually downloading the file.

## [Syntax](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/HEAD#syntax)

HEAD /index.html

# OPTIONS

The **HTTP OPTIONS method** requests permitted communication options for a given URL or server. A client can specify a URL with this method, or an asterisk (\*) to refer to the entire server.

## [Syntax](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/OPTIONS#syntax)

OPTIONS /index.html HTTP/1.1

OPTIONS \* HTTP/1.1

# PATCH

The **HTTP PATCH request method** applies partial modifications to a resource.

PATCH is somewhat analogous to the "update" concept found in [CRUD](https://developer.mozilla.org/en-US/docs/Glossary/CRUD) (in general, HTTP is different than [CRUD](https://developer.mozilla.org/en-US/docs/Glossary/CRUD), and the two should not be confused).

A PATCH request is considered a set of instructions on how to modify a resource. Contrast this with [PUT](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT); which is a complete representation of a resource.

A PATCH is not necessarily idempotent, although it can be. Contrast this with [PUT](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT); which is always idempotent. The word "idempotent" means that any number of repeated, identical requests will leave the resource in the same state. For example if an auto-incrementing counter field is an integral part of the resource, then a [PUT](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT) will naturally overwrite it (since it overwrites everything), but not necessarily so for PATCH.

PATCH (like [POST](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST)) may have side-effects on other resources.

To find out whether a server supports PATCH, a server can advertise its support by adding it to the list in the [Allow](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Allow) or [Access-Control-Allow-Methods](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Methods) (for [CORS](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS)) response headers.

Another (implicit) indication that PATCH is allowed, is the presence of the [Accept-Patch](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Patch) header, which specifies the patch document formats accepted by the server.

## [Syntax](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH#syntax)

PATCH /file.txt HTTP/1.1

# POST

The **HTTP POST method** sends data to the server. The type of the body of the request is indicated by the [Content-Type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type) header.

The difference between [PUT](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT) and POST is that PUT is idempotent: calling it once or several times successively has the same effect (that is no side effect), where successive identical POST may have additional effects, like passing an order several times.

A POST request is typically sent via an [HTML form](https://developer.mozilla.org/en-US/docs/Learn/Forms) and results in a change on the server. In this case, the content type is selected by putting the adequate string in the [enctype](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form" \l "attr-enctype) attribute of the [<form>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form) element or the [formenctype](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input" \l "attr-formenctype) attribute of the [<input>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input) or [<button>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/button) elements:

* application/x-www-form-urlencoded: the keys and values are encoded in key-value tuples separated by '&', with a '=' between the key and the value. Non-alphanumeric characters in both keys and values are [percent encoded](https://developer.mozilla.org/en-US/docs/Glossary/percent-encoding): this is the reason why this type is not suitable to use with binary data (use multipart/form-data instead)
* multipart/form-data: each value is sent as a block of data ("body part"), with a user agent-defined delimiter ("boundary") separating each part. The keys are given in the Content-Disposition header of each part.
* text/plain

When the POST request is sent via a method other than an HTML form — like via an [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) — the body can take any type. As described in the HTTP 1.1 specification, POST is designed to allow a uniform method to cover the following functions:

* Annotation of existing resources
* Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles;
* Adding a new user through a signup modal;
* Providing a block of data, such as the result of submitting a form, to a data-handling process;
* Extending a database through an append operation.

## [Syntax](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST#syntax)

POST /test

POST /test HTTP/1.1

# PUT

The **HTTP PUT request method** creates a new resource or replaces a representation of the target resource with the request payload.

## [Syntax](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT#syntax)

PUT /new.html HTTP/1.1

# TRACE

The **HTTP TRACE method** performs a message loop-back test along the path to the target resource, providing a useful debugging mechanism.

The final recipient of the request should reflect the message received, excluding some fields described below, back to the client as the message body of a [200](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200) (OK) response with a [Content-Type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type) of message/http. The final recipient is either the origin server or the first server to receive a [Max-Forwards](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Max-Forwards) value of 0 in the request.

## [Syntax](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/TRACE#syntax)

TRACE /index.html