

HTTP: Hypertext Transfer Protocol



- [Guides](#)
 - [Security and privacy](#)
 - [Related resources](#)
 - [Tools & resources](#)
 - [Reference](#)
-

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, such as machine-to-machine communication, programmatic access to APIs, and more.

HTTP follows a classical [client-server model](#), with a client opening a connection to make a request, then waiting until it receives a response from the server. HTTP is a [stateless protocol](#), meaning that the server does not keep any session data between two requests, although the later addition of [cookies](#) adds state to some client-server interactions.

Guides

HTTP is an extensible protocol that relies on concepts like resources and Uniform Resource Identifiers (URIs), a basic message structure, and client-server communication model. On top of these concepts, numerous extensions have been developed over the years that add functionality and updated semantics, including additional HTTP methods and headers.

The [HTTP guides](#) are listed in order from general overviews to specialized, use-case-driven topics. Beginners are encouraged to start with the foundational guides before exploring more focused articles.

- [Overview of HTTP](#)

- : The basic features of HTTP, what it can do, its intended use in web architecture, and its position in the protocol stack.
- [Evolution of HTTP](#)
 - : HTTP was created in the early 1990s and has been extended several times. This article goes through its history and describes HTTP/0.9, HTTP/1.0, HTTP/1.1, through HTTP/2 and HTTP/3, as well as novelties introduced over the years.
- [A typical HTTP session](#)
 - : Describes the flow of an HTTP session, from establishing a connection, sending a request, to receiving a response.
- [HTTP messages](#)
 - : HTTP messages transmitted as requests and responses have a defined structure. This article describes this general structure, its purpose, and the different types of messages.
- [MIME types](#)
 - : Since HTTP/1.0, different types of content can be transmitted. This article explains how this is accomplished using the `{{HTTPHeader("Content-Type")}}` header and the MIME standard. A shortlist of common types used by web developers can be found in [Common MIME types](#).
- [Compression in HTTP](#)
 - : Browsers and servers compress their messages before sending them over the network to reduce the amount of data that needs to be transmitted, improving transfer speed and bandwidth utilization.
- [HTTP caching](#)
 - : Caching is a highly important mechanism for delivering fast experiences on the Web and for efficient use of resources. This article describes different methods of caching and how to use HTTP headers to control them.
- [HTTP authentication](#)
 - : Authentication is a way to verify the identity of a client when making requests to a server. It ensures that only authorized users or systems can access certain resources.
- [Using HTTP cookies](#)
 - : Although HTTP is a stateless protocol, a server can send a `{{HTTPHeader("Set-Cookie")}}` header with the response. The client then returns the cookie's value with every subsequent request to the server in the form of a `{{HTTPHeader("Cookie")}}` request header. This adds the ability to store and exchange a small amount of data which effectively adds state to some client-server interactions.
- [Redirections in HTTP](#)
 - : URL redirection, also known as URL forwarding, is a technique to give more than one URL address to a page, a form, a whole website, or a web application. HTTP has a special kind of response, called an HTTP redirect, for this operation.
- [HTTP conditional requests](#)

- : In conditional requests, the outcome of a request depends on the value of a validator in the request. This method is used heavily in [caching](#) and use cases such as resuming a download, preventing lost updates when modifying a document on the server, and more.
- [HTTP range requests](#)
 - : A range request asks the server to send a specific part (or parts) of a resource back to a client instead of the full resource. Range requests are useful for cases when a client knows they need only part of a large file, or for cases where an application allows the user to pause and resume a download.
- [Content negotiation](#)
 - : HTTP defines a set of message headers, starting with [Accept](#) as a way for a browser to announce the format, language, or encoding it prefers. This article explains how this advertisement happens, how the server is expected to react, and how it chooses the most adequate response to a request.
- [Connection management in HTTP/1.x](#)
 - : HTTP/1.1 was the first version of HTTP to support persistent connections and pipelining. This article explains both concepts, including the pros and cons of each.
- [Protocol upgrade mechanism](#)
 - : HTTP/1.1 provides a mechanism to upgrade an already-established connection to a different protocol using the `{{HTTPHeader("Upgrade")}}` header. A client can upgrade a connection from HTTP/1.1 to HTTP/2, or an HTTP(S) connection to a [WebSocket](#) (`ws` / `wss`).
- [Proxy servers and tunneling](#)
 - : A proxy can be on the user's local computer, or anywhere between the user's computer and a destination server on the Internet. This page outlines some basics about proxies and introduces a few configuration options.
- [HTTP Client hints](#)
 - : Client Hints are a set of response headers that a server can use to proactively request information from a client about the device, network, user, and user-agent-specific preferences. The server can then determine which resources to send, based on the information that the client chooses to provide.
- [Network Error Logging](#) `{{experimental_inline}}`
 - : Network Error Logging is a mechanism that can be configured via the [NEL](#) HTTP response header. This experimental header allows websites and applications to opt-in to receive reports about failed (or even successful) network fetches from supporting browsers.
- [Browser detection using the user agent](#)
 - : It's very rarely a good idea to use user agent sniffing to detect a browser, but there are edge cases that require it. This document will guide you in doing this as correctly as possible when this is necessary, with an emphasis on considerations to make before embarking on this route.

Security and privacy

- [Permissions Policy](#)
 - : Permissions Policy provides mechanisms for web developers to explicitly declare what functionality can and cannot be used on a website. You define a set of "policies" that restrict what APIs the site's code can access or modify the browser's default behavior for certain features.
- [Cross-Origin Resource Sharing \(CORS\)](#)
 - : Cross-site HTTP requests are requests for resources from a different domain than that of the resource making the request. Web pages today very commonly load cross-site resources, for example, a page 'Domain A' (`http://domaina.example/`) requests an image on 'Domain B' (`http://domainb.foo/image.jpg`) via the `img` element. CORS allows web developers to control how their site reacts to cross-site requests.
- [Content Security Policy \(CSP\)](#)
 - : CSP allows website administrators to use the `{ {HTTPHeader("Content-Security-Policy") } }` response header to control which resources the client is allowed to load for a given page. The CSP guide describes the overall Content Security Policy mechanism which helps detect and mitigate certain types of attacks, including Cross-Site Scripting (XSS) and data injection attacks.
- [Cross-Origin Resource Policy \(CORP\)](#)
 - : CORP lets websites and applications opt in to protection against specific requests from other origins (such as those issued with elements like `<script>` and `<img>`), to mitigate speculative side-channel attacks.
- [Mozilla web security guidelines](#)
 - : A collection of tips to help operational teams with creating secure web applications.

Related resources

- [URIs](#)
 - : Uniform Resource Identifiers (URIs) are used to describe and locate resources on the web and are an essential component in HTTP requests.
- [Configuring servers for Ogg media](#)
 - : This guide covers a few server configuration changes that may be necessary for your web server to correctly serve Ogg media files. This information may also be useful if you encounter other media types your server isn't already configured to recognize.

Tools & resources

Helpful tools and resources for understanding and debugging HTTP.

- [Firefox Developer Tools](#)
 - : [Network monitor](#)
- [HTTP Observatory](#)
 - : A project designed to help developers, system administrators, and security professionals configure their sites safely and securely.
- [RedBot](#)
 - : Tools to check your cache-related headers.
- [nghttp2](#)
 - : An HTTP/2 client, server and proxy implementation written in C with load test and benchmarking tools and an HPACK encoder and decoder.
- [curl](#)
 - : A command-line tool for transferring data specified with URL syntax. Supports HTTP, HTTPS, WS, WSS, among many other protocols.
- [How Browsers Work \(2011\)](#)
 - : A very comprehensive article on browser internals and request flow through HTTP protocol.

Reference

The [HTTP reference](#) documentation contains detailed information about headers, request methods, status responses, and lists relevant specifications and standards documents.

- [HTTP headers](#)
 - : Message headers are used to send metadata about a resource or an HTTP message, and to describe the behavior of the client or the server.
- [HTTP request methods](#)
 - : Request methods indicate the purpose of the request and what is expected if the request is successful. The most common methods are `{HTTPMethod("GET")}` and `{HTTPMethod("POST")}` for retrieving and sending data to servers, respectively, but there are other methods which serve different purposes.
- [HTTP response status codes](#)
 - : Response status codes indicate the outcome of a specific HTTP request. Responses are grouped in five classes: informational, successful, redirections, client errors, and server errors.
- [HTTP resources and specifications](#)
 - : This page lists relevant resources about HTTP since it was first specified in the early 1990s.

The following subsections are also notable:

- [CSP directives](#)
 - : The `{{HTTPHeader("Content-Security-Policy")}}` (CSP) response header allows website administrators to specify which resources the user agent is allowed to load for a given page. This section lists directives that can be used in a CSP header, with individual documentation pages that describe how the directives work and how to use them.
- [Permissions-Policy directives](#)
 - : The `{{HTTPHeader("Permissions-Policy")}}` response header provides a mechanism to allow or deny the use of browser features in a document or within any `{{HTMLElement("iframe")}}` element in the document. This section lists directives that can be used in a Permissions-Policy header, with individual documentation pages that describe how the directives work and how to use them.

🕒 #2	
🕒 🕒	🕒 2026-10-31 03:27:53 UTC
🕒 🕒	🕒 2026-10-31 03:27:53 UTC