What is a URL?

This article discusses Uniform Resource Locators (URLs), explaining what they are and how they're structured.

Prerequisites:	You need to first know how the Internet works, what a Web server is and the concepts behind links on the web.
Objective:	You will learn what a URL is and how it works on the Web.

Summary

With Hypertext and HTTP, **URL** is one of the key concepts of the Web. It is the mechanism used by browsers to retrieve any published resource on the web.

URL stands for *Uniform Resource Locator*. A URL is nothing more than the address of a given unique resource on the Web. In theory, each valid URL points to a unique resource. Such resources can be an HTML page, a CSS document, an image, etc. In practice, there are some exceptions, the most common being a URL pointing to a resource that no longer exists or that has moved. As the resource represented by the URL and the URL itself are handled by the Web server, it is up to the owner of the web server to carefully manage that resource and its associated URL.

Active Learning

There is no active learning available yet. Please, consider contributing.

Deeper dive

Basics: anatomy of a URL

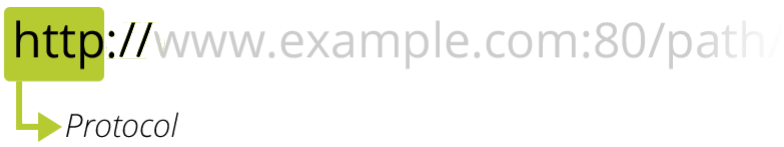
Here are some examples of URLs:

```
https://developer.mozilla.org
https://developer.mozilla.org/en-US/docs/Learn/
https://developer.mozilla.org/en-US/search?q=URL
```

Any of those URLs can be typed into your browser's address bar to tell it to load the associated page (resource).

A URL is composed of different parts, some mandatory and others optional. Let's see the most important parts using the following URL:

```
http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#Som
```

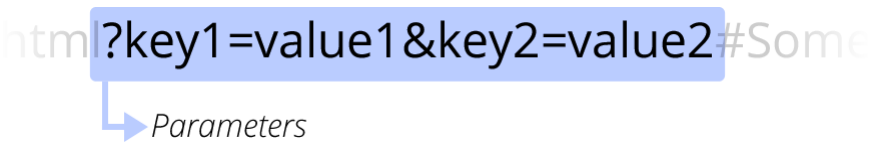


`http` is the protocol. The first part of the URL indicates which protocol the browser must use. A protocol is a set method for exchanging or transferring data around a computer network. Usually for websites it is the HTTP protocol or its secured version, HTTPS. The Web requires one of these two, but browsers also know how to handle other protocols such as `mailto:` (to open a mail client) or `ftp:` to handle file transfer, so don't be surprised if you see such protocols.

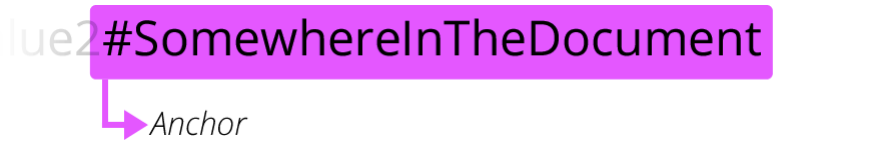
`www.example.com` is the domain name. It indicates which Web server is being requested. Alternatively, it is possible to directly use an IP address, but because it is less convenient, it is not often used on the Web.

`:80` is the port. It indicates the technical "gate" used to access the resources on the web server. It is usually omitted if the web server uses the standard ports of the HTTP protocol (80 for HTTP and 443 for HTTPS) to grant access to its resources. Otherwise it is mandatory.

`/path/to/myfile.html` is the path to the resource on the Web server. In the early days of the Web, a path like this represented a physical file location on the Web server. Nowadays, it is mostly an abstraction handled by Web servers without any physical reality.



`?key1=value1&key2=value2` are extra parameters provided to the Web server. Those parameters are a list of key/value pairs separated with the `&` symbol. The Web server can use those parameters to do extra stuff before returning the resource. Each Web server has its own rules regarding parameters, and the only reliable way to know if a specific Web server is handling parameters is by asking the Web server owner.



`#SomewhereInTheDocument` is an anchor to another part of the resource itself. An anchor represents a sort of "bookmark" inside the resource, giving the browser the directions to show the content located at that "bookmarked" spot. On an HTML document, for example, the browser will scroll to the point where the anchor is defined; on a video or audio

document, the browser will try to go to the time the anchor represents. It is worth noting that the part after the **#**, also known as the ***fragment identifier****, is never sent to the server with the request.*

> **Note:** There are some extra parts and some extra rules regarding URLs, but they are not relevant for regular users or Web developers. Don't worry about this, you don't need to know them to build and use fully functional URLs.

You might think of a URL like a regular postal mail address: the *protocol* represents the postal service you want to use, the *domain name* is the city or town, and the *port* is like the zip code; the *path* represents the building where your mail should be delivered; the *parameters* represent extra information such as the number of the apartment in the building; and, finally, the *anchor* represents the actual person to whom you've addressed your mail.

How to use URLs

Any URL can be typed right inside the browser's address bar to get to the resource behind it. But this is only the tip of the iceberg!

The HTML language — which will be discussed later on — makes extensive use of URLs:

- to create links to other documents with the `<a>` element;
- to link a document with its related resources through various elements such as `<link>` or `<script>`;
- to display medias such as images (with the `<img>` element), videos (with the `<video>` element), sounds and music (with the `<audio>` element), etc.;
- to display other HTML documents with the `<iframe>` element.

> **Note:** When specifying URLs to load resources as part of a page (such as when using the `<script>`, `<audio>`, `<img>`, `<video>`, and the like), you should only use HTTP and HTTPS URLs. Using FTP, for example, is not particularly secure and is no longer supported by many browsers.

Other technologies, such as CSS or JavaScript, use URLs extensively, and these are really the heart of the Web.

Absolute URLs vs relative URLs

What we saw above is called an *absolute URL,* but there is also something called a *relative URL*. Let's examine what that distinction means in more detail.

The required parts of a URL depend to a great extent on the context in which the URL is used. In your browser's address bar, a URL doesn't have any context, so you must provide a full (or *absolute*) URL, like the ones we saw above. You don't need to include the protocol (the browser uses HTTP by default) or the port (which is only required when the targeted Web server is using some unusual port), but all the other parts of the URL are necessary.

When a URL is used within a document, such as in an HTML page, things are a bit different. Because the browser already has the document's own URL, it can use this information to fill in the missing parts of any URL available inside that document. We can differentiate between an *absolute URL* and a *relative URL* by looking only at the *path* part of the URL. If the path part of the URL starts with the " / " character, the browser will fetch that resource from the top root of the server, without reference to the context given by the current document.

Let's look at some examples to make this clearer.

Examples of absolute URLs

Full URL (the same as the one we used before)

```
https://developer.mozilla.org/en-US/docs/Learn
```

Implicit protocol

```
//developer.mozilla.org/en-US/docs/Learn
```

In this case, the browser will call that URL with the same protocol as the one used to load the document hosting that URL.

Implicit domain name

```
/en-US/docs/Learn
```

This is the most common use case for an absolute URL within an HTML document. The browser will use the same protocol and the same domain name as the one used to load the document hosting that URL. **Note:** *it isn't possible to omit the domain name without omitting the protocol as well*.

Examples of relative URLs

To better understand the following examples, let's assume that the URLs are called from within the document located at the following URL: `https://developer.mozilla.org/en-US/docs/Learn`

Sub-resources

```
Skills/Infrastructure/Understanding_URLs
```

Because that URL does not start with `/`, the browser will attempt to find the document in a sub-directory of the one containing the current resource. So in this example, we really want to reach this URL: `https://developer.mozilla.org/en-US/docs/Learn/Skills/Infrastructure/Understanding_URLs`

Going back in the directory tree

```
../CSS/display
```

In this case, we use the `../` writing convention — inherited from the UNIX file system world — to tell the browser we want to go up from one directory. Here we want to reach this URL: `https://developer.mozilla.org/en-US/docs/Learn/../CSS/display`, which can be simplified to: `https://developer.mozilla.org/en-US/docs/CSS/display`

Semantic URLs

Despite their very technical flavor, URLs represent a human-readable entry point for a Web site. They can be memorized, and anyone can enter them into a browser's address bar. People are at the core of the Web, and so it is considered best practice to build what is called *semantic URLs*. Semantic URLs use words with inherent meaning that can be understood by anyone, regardless of their technical know-how.

Linguistic semantics are of course irrelevant to computers. You've probably often seen URLs that look like mashups of random characters. But there are many advantages to creating human-readable URLs:

- It is easier for you to manipulate them.
- It clarifies things for users in terms of where they are, what they're doing, what they're reading or interacting with on the Web.
- Some search engines can use those semantics to improve the classification of the associated pages.

Next steps

- Understanding domain names

Last modified: Mar 23, 2019, by MDN contributors