

# The Web Storage API

## (localStorage, sessionStorage)

### INTRODUCTION

The Web storage API (see the [related W3C specification](#)) introduces "two related mechanisms, similar to HTTP session cookies, for storing structured data on the client side".



Indeed, Web Storage provides two interfaces called `sessionStorage` and `localStorage`, whose main difference is data longevity. This specification defines an API for persistent data storage of key-value pair data in Web clients.

**With `localStorage` the data will remain until it is deleted, whereas with `sessionStorage` the data is erased when the tab/browser is closed.**

For convenience, we will mainly illustrate the `localStorage` object. Just change "local" by "session" and it should work (this time with a session lifetime).

SIMPLE KEY-VALUE STORES, ONE PER DOMAIN (FOLLOWING THE [SAME ORIGIN POLICY](#))!

`localStorage` is a simple key-value store, in which the keys and values are strings. There is only one store per domain. This functionality is exposed through the globally available `localStorage` object. The same applies to `sessionStorage`.

key	value
firstName	Bugs
lastName	Bunny
location	Earth

Example:

```
// Using localStorage
// store data
localStorage.lastName = "Bunny";
localStorage.firstName = "Bugs";
localStorage.location = "Earth";
// retrieve data
var lastName = localStorage.lastName;
10. var firstName = localStorage.firstName;
var location = localStorage.location;
```

This data is located in a store attached to the origin of the page. We created [a JS Bin example in which we included the above code](#).

Once opened in your browser, the JavaScript code is executed. With the browser dev. tools, we can check what has been stored in the `localStorage` for this domain:

Browser window showing a JS Bin editor interface. The address bar displays `jsbin.com/taxono/e...`. The interface includes tabs for HTML, CSS, JavaScript, Console, and Output. The HTML tab is active, showing the following code:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>JS Bin</title>
  <script>
    // Using localStorage
    // store data
    localStorage.lastName = "Bunny";
    localStorage.firstName = "Bugs";
    localStorage.location = "Earth";

    // retrieve data
    var lastName =
localStorage.lastName;
    var firstName =
localStorage.firstName;
    var location =
localStorage.location;

    console.log("lastName just retrieved
from the localStorage of your browser,
for this particular web site: " +
lastName);
  </script>
</head>
```

The Output tab is also active, showing a red pixelated graphic. Two red arrows point from the code to the output:

- One arrow points from the code `localStorage.lastName = "Bunny";` to the output, with the text: **This code writes to localStorage**
- Another arrow points from the code `var lastName = localStorage.lastName;` to the output, with the text: **This code reads from localStorage**

The screenshot shows a web browser window with the URL `jsbin.com/povuqa/1/edit?html`. The HTML editor displays the following code:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>JS Bin</title>
  <script>
    // Using localStorage
    // store data
    localStorage.lastName = "Bunny";
    localStorage.firstName = "Bugs";
    localStorage.location = "Earth";

    // retrieve data
    var lastName = localStorage.lastName;
    var firstName = localStorage.firstName;
```

Red arrows point from the text **These instructions created these data!** to the JavaScript code. The browser's developer tools are open to the **Resources** tab, which shows a table of data:

Key	Value
firstName	Bugs
lastName	Bunny
location	Earth

Below the table, the **Local Storage** section is expanded, showing two entries:

- `http://jsbin.com`
- `http://null.jsbin.com` (highlighted with a red box)

Red arrows point from the text **url of jsbin in standalone mode** to `http://jsbin.com` and from **url of jsbin in edit mode** to `http://null.jsbin.com`.

## DIFFERENCES WITH COOKIES?

Cookies are also a popular way to store key-value pairs. Web Storage however, is a more powerful technique than cookies. The latter are limited in size (a few KBytes for cookies (compared to several MBytes for Web Storage) and they generate HTTP traffic for each additional request (whether to request a Web

page, an image, a stylesheet, a JavaScript file, etc.).

Objects managed by Web Storage are no longer carried on the network and HTTP, and are easily accessible (read, change and delete) from JavaScript, using the Web Storage API.

DO NOT TRY WITH `FILE://`

**WARNING!** `localStorage` and `sessionStorage` are not usable with `file://`, only with `http://` or `https://`

For security reasons (more on that later), pages loaded with a `file://` type of URL cannot use `localStorage` or `sessionStorage`. You must use `http://` or `https://` URLs and a Web server.

## EXTERNAL RESOURCES

- [The W3C Web Storage API recommendation \(published on 9 June 2015\)](#)
- [Interesting article on html5rocks that compares the different ways of doing client side persistence with HTML5, including Web Storage.](#)