Pocket Guides  Go Make Things

5

BROWSER STORAGE

with Vanilla JavaScript

CHRIS FERDINANDI

Browser Storage

By Chris Ferdinandi
Go Make Things, LLC
v2.1.0

Table of Contents

1. Intro
2. Cookies
3. Storing Data in the Browser
4. When to use cookies vs. localStorage
5. Query Strings
6. Get an individual query string value
7. Putting it all together
8. About the Author

Intro

In this guide, you'll learn:

- How to set, get, and remove cookies.
- How to store data in the browser with localStorage and sessionStorage.
- How to get data from query strings.
- How to put it all together and write a working project with browser storage.

A quick word about browser compatibility

This guide makes heavy use of ECMAScript 5 (more commonly known as ES5) and ECMA 6 (ES6) methods and APIs.

My goal for browser support is IE9 and above. Each function or technique mentioned in this guide includes specific browser support information. For methods and APIs that don't meet that standard, I also include information about polyfills—snippets of code that add support for features to browsers that don't natively offer it.

You'll never have to run a command line prompt, compile code, or learn a weird pseudo language (though you certain can if you want to).

Note: *You can extend support all the way back to IE7 with a polyfill service like polyfill.io*[1].

Using the code in this guide

Let's get started!

Cookies

How to get, set, and remove cookies.

Setting a cookie

You can use `document.cookie` to set a cookie. It's a string, using a `{KEY}={VALUE};` format. Optionally, you can pass in an expiration date as a timestamp using the `expires={VALUE}` format.

```
// Set a cookie named sandwich, with a value of turkey
// Cookie expires on December 31, 2024 at 11:59 and 59 seconds PM
document.cookie = 'sandwich=turkey; expires=Fri, 31 Dec 2024 23:59:59 GMT';
```

Browser Compatibility

Works in all modern browsers, and at least IE6.

Getting a cookie value

Getting a cookie value involves parsing a string, and can be a bit awkward to work with. `getCookie()` is a super lightweight helper method[2](https://github.com/cferdinandi/vanilla-javascript-cheat-sheet/blob/master/helper-methods/getCookie.js) to make this easier.

Here's how it works.

```
var cookieVal = getCookie('sandwich');
```

And here's the helper method.

```
/**
 * Get the value of a cookie
 * Source: https://gist.github.com/wpsmith/6cf23551dd140fb72ae7
 * @param  {String} name  The name of the cookie
 * @return {String}       The cookie value
 */
var getCookie = function (name) {
    var value = "; " + document.cookie;
    var parts = value.split("; " + name + "=");
    if (parts.length == 2) return parts.pop().split(";").shift(
);
};
```

Browser Compatibility

Works in all modern browsers, and at least IE6.

More complex cookies

If you're doing more complex work with cookies, I would strongly recommend the simple cookie library provided by MDN.[3]. It let's you easily set, get, and remove cookies.

```
// Set a cookie
docCookies.setItem('sandwich', 'turkey with tomato and mayo', n
ew Date(2020, 5, 12));

// Get a cookie
var cookieVal = docCookies.getItem('sandwich');

// Remove a cooookie
docCookies.removeItem('sandwich');
```

Browser Compatibility

Works in all modern browsers, and at least IE6.

Storing Data in the Browser

How to store data locally in the browser.

localStorage

Store data locally that the browser can access later. Stored indefinitely. Stored data must be a string.

```
// Store data
var someData = 'The data that I want to store for later.';
localStorage.setItem('myDataKey', someData);

// Get data
var data = localStorage.getItem('myDataKey');

// Remove data
localStorage.removeItem('myDatakey');
```

Browser Compatibility

Works in all modern browsers, and IE8 and above.

sessionStorage

Session storage works just like `localStorage`, except the data is cleared when the browser session ends.

```
// Store data
var someTempData = 'The data that I want to store temporarily.'
;
sessionStorage.setItem('myTempDataKey', someTempData);

// Get data
var tempData = sessionStorage.getItem('myTempDataKey');

// Remove data
sessionStorage.removeItem('myTempDatakey');
```

Browser Compatibility

Works in all modern browsers, and IE8 and above.

Storage Limits

Browsers provide differing levels of storage space for `localStorage` and `sessionStorage`,[4] ranging from as little as 2mb up to unlimited.

For browsers with a maximum storage limit, this amount is a total allowable amount of data, not *just* a max for your specific site or web app. Accordingly, you should try to reduce the overall footprint of your data as much as possible.

Storing arrays and objects

While `localStorage` and `sessionStorage` can only data in string form, you can convert arrays and objects to strings (and then transform them back). This allows you to store multiple values as a single item, reducing your overall data

footprint.

Arrays

For arrays, we'll use the `toString()` method to convert the array to a string, and `split()` to convert it back to an array.

```
var someArray = ['turkey', 'tuna', 'pb&j'];

// Save data
localStorage.setItem('sandwiches', someArray.toString());

// Get data
var data = localStorage.getItem('sandwiches').split(',');
```

Objects

For objects, we'll use `JSON.stringify()` to convert our object to a JSON string, and `JSON.parse` to convert it back.

```
var lunch = {
    sandwich: 'turkey',
    chips: 'cape cod',
    drink: 'soda'
}

// Save data
localStorage.setItem('lunch', JSON.stringify(lunch));

// Get data
var data = JSON.parse(localStorage.getItem('lunch'));
```

Browser Compatibility

Works in all modern browsers, and IE8 and above.

When to use cookies vs. localStorage

When should you use cookies, and when should you use `localStorage`? There's no one right answer, but there are some general guidelines you might want to follow.

Use cookies...

- For small pieces of data. They have broader browser support and don't use the browser's maximum allowed `localStorage` quota.
- If your data needs to expire on a specific date or time, since `localStorage` has no expiration setting beyond `sessionStorage`.
- If you need to access the data on the server. Most server-side languages have access to stored cookie data.

Use `localStorage`...

- For larger amounts of data, since cookies max out at 4kb in size.
- For data you want to easily expire when the session ends, since that functionality is baked right in to `sessionStorage`.

Query Strings

Get query string values from a URL.

Get an individual query string value

`getQueryString` is a helper method[5] I wrote to get the value of a query string from a URL. Pass in the key to get the value of. You can optionally pass in a URL as a second argument. The function will use the window URL by default.

Here's how it works.

```
// Sample URL: http://example.com?sandwich=turkey&snack=cookies
var sandwich = getQueryString('sandwich'); // returns 'turkey'
var snack = getQueryString('snack'); // returns 'cookies'
var dessert = getQueryString('dessert'); // returns null
var drink = getQueryString('drink', 'http://another-example.com
?drink=soda'); // returns 'soda'
```

And here's the helper method.

```
/*!
 * Get the value of a query string from a URL
 * (c) 2017 Chris Ferdinandi, MIT License, https://gomakethings
.com
 * @param  {String} field The field to get the value of
 * @param  {String} url   The URL to get the value from [option
al]
 * @return {String}       The value
 */
var getQueryString = function (field, url) {
    var href = url ? url : window.location.href;
    var reg = new RegExp('[?&]' + field + '=([^&#]*)', 'i');
    var string = reg.exec(href);
    return string ? string[1] : null;
};
```

Browser Compatibility

Works in all modern browsers, and at least IE6.

Get all query string parameters

`getParams()` is a helper method[6] adapted from CSS Tricks that returns an object containing key/value pairs for all of the query strings in a URL. Pass in the URL as an argument.

Here's how it works.

```
// Get query strings from the current URL
var params = getParams(window.location.href);

// Get query strings from any string
var moreParams = getParams('http://example.com?q=sandwich&type=
tuna&sauce=mayo&topping=tomato%20and%20lettuce');
```

And here's the helper method.

```
/**
 * Get the URL parameters
 * source: https://css-tricks.com/snippets/javascript/get-url-v
ariables/
 * @param  {String} url The URL
 * @return {Object}     The URL parameters
 */
var getParams = function (url) {
    var params = {};
    var parser = document.createElement('a');
    parser.href = url;
    var query = parser.search.substring(1);
    var vars = query.split('&');
    for (var i=0; i < vars.length; i++) {
        var pair = vars[i].split("=");
        params[pair[0]] = decodeURIComponent(pair[1]);
    }
    return params;
};
```

Browser Compatibility

Works in all modern browsers, and at least IE6.

Putting it all together

To make this all tangible, let's work on a project together. We'll build a script that automatically saves form content whenever it's updated.

The starter template and complete project code are included in the source code[7] on GitHub.

Getting Setup

First, let's create a few simple fields we can enter data into.

```
<div>
    <label for="name">Name</label>
    <input type="text" id="name">
</div>

<div>
    <label for="email">Email Address</label>
    <input type="email" id="email">
</div>

<div>
    <label for="notes">Notes</label>
    <textarea id="notes"></textarea>
</div>
```

Now we're ready to write our script.

Listening for changes on the inputs

The `input` event triggers whenever a change is made to an `input`, `textarea`, or `select` field value. That's perfect for what we're trying to do.

Let's setup an event listener for the `input` event.

```
document.addEventListener('input', function (event) {
    // Our code will go here...
}, false);
```

Whenever our event is trigger, we want to save the field's `value` in `localStorage`.

We can use `event.target` to get the field that triggered the event. We'll use `formAutosaveData_` as our base `localStorage` key name, and append the field ID to the end of it. We'll save the `event.target.value` as our value.

```
document.addEventListener('input', function (event) {
    localStorage.setItem('formAutosaveData_' + event.target.id,
 event.target.value);
}, false);
```

And just like that, you're saving form data to `localStorage` whenever it changes.

Loading saved data

Saving data is the easy part. Now let's look at how to load saved data back into the form when the page is loaded.

First, we want to grab all of our form fields. We'll do that using `querySelectorAll` to get all `input` and `textarea` elements on the page.

```
var inputs = document.querySelectorAll('input, textarea');
```

Next, we want to loop through each element and check to see if there's an item in `localStorage` for that field. If there's no saved data, we'll skip to the next item. Otherwise, we'll update our field value with the `localStorage` data.

```
var inputs = document.querySelectorAll('input, textarea');
for (var i = 0; i < inputs.length; i++) {
    var saved = localStorage.getItem('formAutosaveData_' + inpu
ts[i].id);
    if (!saved) continue;
    inputs[i].value = saved;
}
```

Actually, that was pretty easy, too!

Saving all fields as a single entry

If you're working with a large form, you could pretty quickly clutter up `localStorage` with a ton of entries. Ideally, you want to save all of your data as a single entry, using a JavaScript object with the field ID as a key.

```
var data = {
    name: 'Some value',
    email: 'Another value',
    notes: 'A third value'
};
```

Saving our data

For this approach, we need to make a few changes. In our event listener, we want to look up any existing data in `localStorage` so we can add to it.

If data is already saved, we'll convert it from a string to an object with `JSON.parse()` (remember, `localStorage` values can only be strings). If not, we'll create a new object to push our data to.

Below, I'm using a ternary operator as a shorthand if/else check. The content before the `?` is the statement to evaluate. If true, the variable is set to the value between the `?` and `:`. Otherwise, it's set to the value after the `:`.

```
document.addEventListener('input', function (event) {
    var saved = localStorage.getItem('formAutosaveData');
    var data = saved ? JSON.parse(saved) : {};
}, false);
```

Now that we have our object, we want to push our field value to it. We'll use the `event.target.id` to add a new entry or replace an existing one in our `data` variable. Then, we'll convert our object to a string with `JSON.stringify()` and save it to `localStorage`.

```
document.addEventListener('input', function (event) {
    var saved = localStorage.getItem('formAutosaveData');
    var data = saved ? JSON.parse(saved) : {};
    data[event.target.id] = event.target.value;
    localStorage.setItem('formAutosaveData', JSON.stringify(dat
a));
}, false);
```

Loading our data

Now let's look at how to load our saved data onto the page.

Rather than get all of our inputs, we're going to first grab our saved data from `localStorage`. If there's any saved data, we'll convert it to an object with `JSON.parse()`. Then, we'll loop through.

```
var saved = localStorage.getItem('formAutosaveData');
if (saved) {
    saved = JSON.parse(saved);
    for (var fieldId in saved) {
        if (obj.hasOwnProperty(saved, fieldId)) {
            // Do something with our data       }
    }
}
```

For each saved item, we want to find the field the data belongs to. We can use the data key, which is the field's ID, to search for it. If a matching field is found, we'll update it's value.

```
var saved = localStorage.getItem('formAutosaveData');
if (saved) {
    saved = JSON.parse(saved);
    for (var fieldId in saved) {
        if (obj.hasOwnProperty(saved, fieldId)) {
            var field = document.querySelector('#' + fieldId);
            if (!field) continue;
            field.value = saved[fieldId];
        }
    }
}
```

Congratulations! You just created a project with `localStorage`.

About the Author

Hi, I'm Chris Ferdinandi. I help people learn JavaScript.

I love pirates, puppies, and Pixar movies, and live near horse farms in rural Massachusetts. I run Go Make Things with Bailey Puppy, a lab-mix from Tennessee.

You can find me:

- On my website at GoMakeThings.com.
- By email at chris@gomakethings.com.
- On Twitter at @ChrisFerdinandi.

1. https://polyfill.io↩
2. https://github.com/cferdinandi/vanilla-javascript-cheat-sheet/blob/master/helper-methods/getCookie.js↩
3. https://developer.mozilla.org/en-US/docs/Web/API/Document/cookie/Simple_document.cookie_framework↩
4. https://www.html5rocks.com/en/tutorials/offline/quota-research/↩
5. https://github.com/cferdinandi/vanilla-javascript-cheat-sheet/blob/master/helper-methods/getQueryString.js↩
6. https://github.com/cferdinandi/vanilla-javascript-cheat-sheet/blob/master/helper-methods/getParams.js↩
7. https://github.com/cferdinandi/data-storage-source-code↩