Quick Tip: Get URL Parameters with JavaScript

By Yaphi Berhanu, James Hibbard

JavaScript

January 13, 2020

Share:

URL parameters (also called query string parameters or URL variables) are used to send small amounts of data from page to page, or from client to server via a URL. They can contain all kinds of useful information, such as search queries, link referrals, product information, user preferences, and more.

In this article, we'll show you how to parse and manipulate URL parameters using JavaScript.

Getting a URL Parameter

In modern browsers, this has become a lot easier, thanks to the URLSearchParams interface. This defines a host of utility methods to work with the query string of a URL.

Assuming that our URL is `https://example.com/?product=shirt&color=blue&newuser&size=m`, we can grab the query string using `window.location.search`:

```
const queryString = window.location.search;
console.log(queryString);
// ?product=shirt&color=blue&newuser&size=m
```

We can then parse the query string's parameters using `URLSearchParams`:

```
const urlParams = new URLSearchParams(queryString);
```

Then we call any of its methods on the result.

For example, `URLSearchParams.get()` will return the first value associated with the given search parameter:

```
const product = urlParams.get('product')
console.log(product);
// shirt

const color = urlParams.get('color')
console.log(color);
// blue

const newUser = urlParams.get('newuser')
console.log(newUser);
// empty string
```

Other Useful Methods

Checking for the Presence of a Parameter

You can use `URLSearchParams.has()` to check whether a certain parameter exists:

```
console.log(urlParams.has('product'));
// true

console.log(urlParams.has('paymentmethod'));
// false
```

Getting All of a Parameter's Values

You can use `URLSearchParams.getAll()` to return all of the values associated with a particular parameter:

```
console.log(urlParams.getAll('size'));
// [ 'm' ]

//Programmatically add a second size parameter.
urlParams.append('size', 'xl');

console.log(urlParams.getAll('size'));
// [ 'm', 'xl' ]
```

Iterating over Parameters

`URLSearchParams` also provides some familiar `Object` iterator methods, allowing you iterate over its keys, values and entries:

```
const
  keys = urlParams.keys(),
  values = urlParams.values(),
  entries = urlParams.entries();

for (const key of keys) console.log(key);
// product, color, newuser, size

for (const value of values) console.log(value);
```

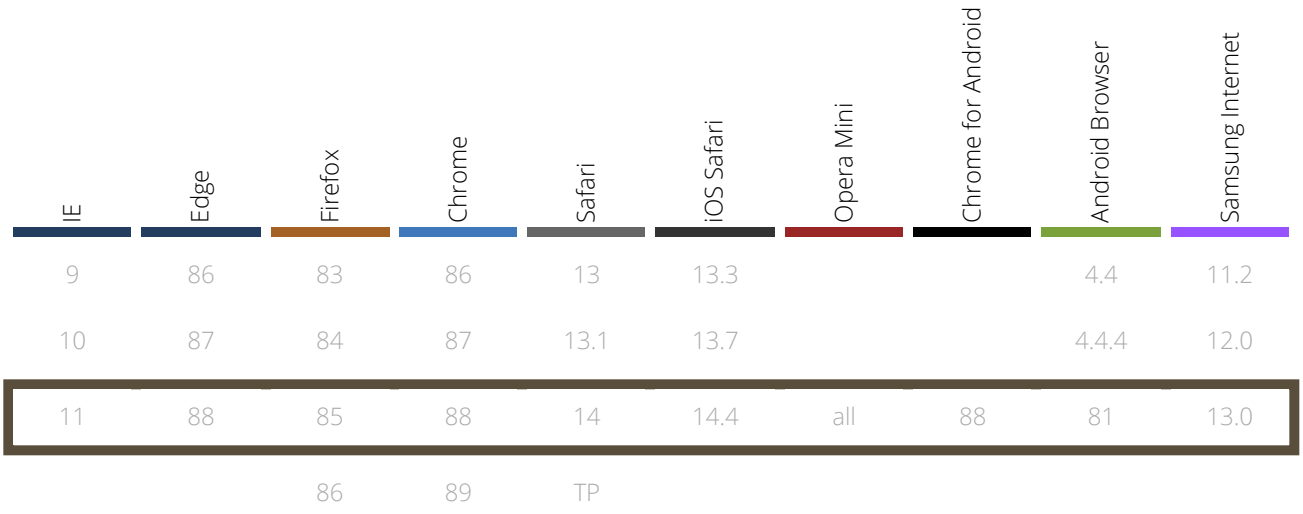
```
for(const entry of entries) {
  console.log(`${entry[0]}: ${entry[1]}`);
}
// product: shirt
// color: blue
// newuser:
// size: m
```

Browser Support

Browser support for `URLSearchParams` is good. At the time of writing, it's supported in all major browsers.

URLSearchParams

The URLSearchParams interface defines utility methods to work with the query string of a URL.



There's a polyfill available if you have to support legacy browsers such as Internet Explorer. *Or*, you could follow along with the rest of this tutorial and learn how to roll your own.

Rolling Your Own Query String Parsing Function

Let's stay with the URL we were using in the previous section:

```
http://example.com/?product=shirt&color=blue&newuser&size=m
```

Here's a function to give you all the URL parameters as a neat object:

```
function getAllUrlParams(url) {

  // get query string from url (optional) or window
  var queryString = url ? url.split('?')[1] : window.location.search.slice(

  // we'll store the parameters here
  var obj = {};

  // if query string exists
  if (queryString) {

    // stuff after # is not part of query string, so get rid of it
    queryString = queryString.split('#')[0];

    // split our query string into its component parts
    var arr = queryString.split('&');

    for (var i = 0; i < arr.length; i++) {
      // separate the keys and the values
      var a = arr[i].split('=');

      // set parameter name and value (use 'true' if empty)
      var paramName = a[0];
      var paramValue = typeof (a[1]) === 'undefined' ? true : a[1];

      // (optional) keep case consistent
      paramName = paramName.toLowerCase();
      if (typeof paramValue === 'string') paramValue = paramValue.toLowerCa
```

```
      // if the paramName ends with square brackets, e.g. colors[] or color
      if (paramName.match(/\[(\d+)?\]$/)) {

        // create key if it doesn't exist
        var key = paramName.replace(/\[(\d+)?\]/, '');
        if (!obj[key]) obj[key] = [];

        // if it's an indexed array e.g. colors[2]
        if (paramName.match(/\[\d+\]$/)) {
          // get the index value and add the entry at the appropriate posit
          var index = /\[(\d+)\]/.exec(paramName)[1];
          obj[key][index] = paramValue;
        } else {
          // otherwise add the value to the end of the array
          obj[key].push(paramValue);
        }
      } else {
        // we're dealing with a string
        if (!obj[paramName]) {
          // if it doesn't exist, create property
          obj[paramName] = paramValue;
        } else if (obj[paramName] && typeof obj[paramName] === 'string'){
          // if property does exist and it's a string, convert it to an arr
          obj[paramName] = [obj[paramName]];
          obj[paramName].push(paramValue);
        } else {
          // otherwise add the property
          obj[paramName].push(paramValue);
        }
      }
    }
  }

  return obj;
}
```

You'll see how this works soon, but first, here are some usage examples:

```
getAllUrlParams().product; // 'shirt'
getAllUrlParams().color; // 'blue'
getAllUrlParams().newuser; // true
getAllUrlParams().nonexistent; // undefined
getAllUrlParams('http://test.com/?a=abc').a; // 'abc'
```

And here's a demo for you to play around with.

Things to Know Before Using This Function

- Our function assumes the parameters are separated by the `&` character, as indicated in the W3C specifications. However, the URL parameter format in general is not clearly defined, so you occasionally might see `;` or `&` as separators.

- Our function still works if a parameter doesn't have an equals sign or if it has an equals sign but no value.

- The values of duplicate parameters get put into an array.

If you just wanted a function you could drop into your code, you're done now. If you'd like to understand how the function works, read on.

The following section assumes you know some JavaScript, including functions, objects, and arrays. If you need a refresher, check out the MDN JavaScript reference.

How the Function Works

Overall, the function takes a URL's query string (the part after the `?` and before the `#`) and spits out the data in a neat object.

First, this line says, if we've specified a URL, get everything after the question mark, but otherwise, just use the URL of the window:

```
var queryString = url ? url.split('?')[1] : window.location.search.slice(1)
```

Next, we'll create an object to store our parameters:

```
var obj = {};
```

If the query string exists, we'll start parsing it. First we have to make sure to shave off the part starting from the `#`, since it's not part of the query string:

```
queryString = queryString.split('#')[0];
```

Now we can split the query string into its component parts:

```
var arr = queryString.split('&');
```

That will give us an array that looks like this:

```
['product=shirt', 'color=blue', 'newuser', 'size=m']
```

Next, we'll loop through this array and split each item into a key and a value, which we'll soon put into our object:

```
var a = arr[i].split('=');
```

Let's assign the key and a value to individual variables. If there isn't a parameter value, we'll set it to `true` to indicate that the parameter name exists. Feel free to change this depending on your use case:

```
var paramName = a[0];
var paramValue = typeof (a[1]) === 'undefined' ? true : a[1];
```

Optionally, you can set all parameter names and values to lowercase. That way, you can avoid situations where someone sends traffic to a URL with `example=TRUE` instead of `example=true` and your script breaks. (I've seen this happen.) However, if your query string needs to be case sensitive, feel free to omit this part:

```
paramName = paramName.toLowerCase();
if (typeof paramValue === 'string') paramValue = paramValue.toLowerCase();
```

Next, we need to deal with the various types of input we can receive in `paramName`. This could be an indexed array, a non-indexed array, or a regular string.

If it's an indexed array, we want the corresponding `paramValue` to be an array, with the value inserted at the correct position. If it's a non-indexed array, we want the corresponding `paramValue` to be an array with the element pushed on to it. If it's a string, we want to create a regular property on the object and assign the `paramValue` to it, *unless* the property already exists, in which case we want to convert the existing `paramValue` to an array and push the incoming `paramValue` on to that.

To illustrate this, here's some sample input, with the output we would expect:

```
getAllUrlParams('http://example.com/?colors[0]=red&colors[2]=green&colors[6
// { "colors": [ "red", null, "green", null, null, null, "blue" ] }

getAllUrlParams('http://example.com/?colors[]=red&colors[]=green&colors[]=b
// { "colors": [ "red", "green", "blue" ] }

getAllUrlParams('http://example.com/?colors=red&colors=green&colors=blue');
// { "colors": [ "red", "green", "blue" ] }

getAllUrlParams('http://example.com/?product=shirt&color=blue&newuser&size=
// { "product": "shirt", "color": "blue", "newuser": true, "size": "m" }
```

And here's the code to implement the functionality:

```
if (paramName.match(/\[(\d+)?\]$/)) {
  var key = paramName.replace(/\[(\d+)?\]/, '');
  if (!obj[key]) obj[key] = [];

  if (paramName.match(/\[\d+\]$/)) {
```

```
      obj[key][index] = paramValue;
    } else {
      obj[key].push(paramValue);
    }
  } else {
    if (!obj[paramName]) {
      obj[paramName] = paramValue;
    } else if (obj[paramName] && typeof obj[paramName] === 'string'){
      obj[paramName] = [obj[paramName]];
      obj[paramName].push(paramValue);
    } else {
      obj[paramName].push(paramValue);
    }
  }
```

Finally, we return our object with the parameters and values.

If your URL has any encoded special characters like spaces (encoded as `%20`), you can also decode them to get the original value like this:

```
// assume a url parameter of test=a%20space

var original = getAllUrlParams().test; // 'a%20space'
var decoded = decodeURIComponent(original); // 'a space'
```

Just be careful not to decode something that's already decoded or else your script will error out, especially if percents are involved.

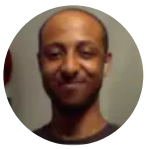
Anyways, congrats! Now you know how to get a URL parameter, and hopefully have picked up some other tricks along the way.

Conclusion

The code in this article works for the most common use cases where you would get a URL query parameter. If you're working with any edge cases, such as uncommon separators or special formatting, then be sure to test and adjust accordingly.

If you want to do more with URLs, there are specific libraries available, such as query-string and Medialize URI.js. But since it's basically string manipulation, it often makes sense just to use plain JavaScript. Whether you use your own code or go with a library, be sure to check everything and make sure it works for your use cases.

This article was updated in 2020 for relevance and accuracy. For more in-depth JavaScript knowledge, read our book, JavaScript: Novice to Ninja, 2nd Edition.



Yaphi Berhanu

Yaphi Berhanu is a web developer who loves helping people boost their coding skills. He writes tips and tricks at http://simplestepscode.com. In his completely unbiased opinion, he suggests checking it out.

James Hibbard

Currently I work for SitePoint as editor of their JavaScript hubs and technical editor for various books (e.g. JavaScript: Novice to Ninja and Jump Start Vue.js). I also work as a network admin and freelance web dev, where I spend a fair bit of my time working on Rails apps.