CSS Variables: Why Should You Care?

By Rob Dodson
Rob is a contributor to Web**Fundamentals**

CSS variables, more accurately known as CSS custom properties, are landing in Chrome 49. They can be useful for reducing repetition in CSS, and also for powerful runtime effects like theme switching and potentially extending/polyfilling future CSS features.

CSS clutter

When designing an application it’s a common practice to set aside a set of brand colors that will be reused to keep the look of the app consistent. Unfortunately, repeating these color values over and over again in your CSS is not only a chore, but also error prone. If, at some point, one of the colors needs to be changed, you could throw caution to the wind and “find-and-replace” all the things, but on a large enough project this could easily get dangerous.

In recent times many developers have turned to CSS preprocessors like SASS or LESS which solve this problem through the use of preprocessor variables. While these tools have boosted developer productivity immensely, the variables that they use suffer from a major drawback, which is that they’re static and can’t be changed at runtime. Adding the ability to change variables at runtime not only opens the door to things like dynamic application theming, but also has major ramifications for responsive design and the potential to polyfill future CSS features. With the release of Chrome 49, these abilities are now available in the form of CSS custom properties.

Custom properties in a nutshell

Custom properties add two new features to our CSS toolbox:

- The ability for an author to assign arbitrary values to a property with an author-chosen name.
- The `var()` function, which allows an author to use these values in other properties.

Here’s a quick example to demonstrate

```
:root {
  --main-color: #06c;
}

#foo h1 {
  color: var(--main-color);
}
```

`--main-color` is an author defined custom property with a value of #06c. Note that all custom properties begin with two dashes.

The `var()` function retrieves and replaces itself with the custom property value, resulting in `color: #06c;` So long as the custom property is defined somewhere in your stylesheet it should be available to the `var` function.

The syntax may look a little strange at first. Many developers ask, "Why not just use `$foo` for variable names?" The approach was specifically chosen to be as flexible as possible and potentially allow for `$foo` macros in the future. For the backstory you can read this post from one of the spec authors, Tab Atkins.

Custom property syntax

The syntax for a custom property is straightforward.

```
--header-color: #06c;
```

Note that custom properties are case sensitive, so `--header-color` and `--Header-Color` are different custom properties. While they may seem simple at face value, the allowed syntax for custom properties is actually quite permissive. For example, the following is a valid custom property:

```
--foo: if(x > 5) this.width = 10;
```

While this would not be useful as a variable, as it would be invalid in any normal property, it could potentially be read and acted upon with JavaScript at runtime. This means custom properties have the potential to unlock all kinds of interesting techniques not currently possible with today's CSS preprocessors. So if you're thinking "*yawn* I have SASS so who cares…" then take a second look! These are not the variables you're used to working with.

The cascade

Custom properties follow standard cascade rules, so you can define the same property at different levels of specificity

```
:root { --color: blue; }
div { --color: green; }
#alert { --color: red; }
* { color: var(--color); }

<p>I inherited blue from the root element!</p>
<div>I got green set directly on me!</div>
<div id="alert">
  While I got red set directly on me!
  <p>I’m red too, because of inheritance!</p>
</div>
```

This means you can leverage custom properties inside of media queries to aid with responsive design. One use case might be to expand the margining around your major sectioning elements as the screen size increases:

```
:root {
  --gutter: 4px;
}

section {
  margin: var(--gutter);
}

@media (min-width: 600px) {
  :root {
    --gutter: 16px;
  }
}
```

It’s important to call out that the above snippet of code is not possible using today’s CSS preprocessors which are unable to define variables inside of media queries. Having this ability unlocks a lot of potential!

It’s also possible to have custom properties that derive their value from other custom properties. This can be extremely useful for theming:

```
:root {
  --primary-color: red;
  --logo-text: var(--primary-color);
}
```

The var() function

To retrieve and use the value of a custom property you'll need to use the `var()` function. The syntax for the `var()` function looks like this:

```
var(<custom-property-name> [, <declaration-value> ]? )
```

Where `<custom-property-name>` is the name of an author defined custom property, like `--foo`, and `<declaration-value>` is a fallback value to be used when the referenced custom property is invalid. Fallback values can be a comma separated list, which will be combined into a single value. For example `var(--font-stack, "Roboto", "Helvetica");` defines a fallback of `"Roboto", "Helvetica"`. Keep in mind that shorthand values, like those used for margin and padding, are not comma separated, so an appropriate fallback for padding would look like this.

```
p {
  padding: var(--pad, 10px 15px 20px);
}
```

Using these fallback values, a component author can write defensive styles for their element:

```
/* In the component’s style: */
.component .header {
  color: var(--header-color, blue);
}
.component .text {
  color: var(--text-color, black);
}

/* In the larger application’s style: */
.component {
  --text-color: #080;
  /* header-color isn’t set,
     and so remains blue,
     the fallback value */
}
```

This technique is especially useful for theming Web Components that use Shadow DOM, as custom properties can traverse shadow boundaries. A Web Component author can create an initial design using fallback values, and expose theming “hooks” in the form of custom properties.

```
<!-- In the web component's definition: -->
<x-foo>
  #shadow
    <style>
      p {
        background-color: var(--text-background, blue);
      }
    </style>
    <p>
      This text has a yellow background because the document styled me! Otherwise
      would be blue.
    </p>
</x-foo>

/* In the larger application's style: */
x-foo {
  --text-background: yellow;
}
```

When using `var()` there are a few gotchas to watch out for. Variables cannot be property names. For instance:

```
.foo {
  --side: margin-top;
  var(--side): 20px;
}
```

However this is not equivalent to setting `margin-top: 20px;`. Instead, the second declaration is invalid and is thrown out as an error.

Similarly, you can’t (naively) build up a value where part of it is provided by a variable:

```
.foo {
  --gap: 20;
  margin-top: var(--gap)px;
}
```

Again, this is not equivalent to setting `margin-top: 20px;`. To build up a value, you need something else: the `calc()` function.

Building values with calc()

If you’ve never worked with it before, the `calc()` function is a handly little tool that lets you perform calculations to determine CSS values. It’s supported on all modern browsers, and can be combined with custom properties to build up new values. For example:

```
.foo {
  --gap: 20;
  margin-top: calc(var(--gap) * 1px); /* niiiiice */
}
```

Working with custom properties in JavaScript

To get the value of a custom property at runtime, use the `getPropertyValue()` method of the computed CSSStyleDeclaration object.

```
/* CSS */
:root {
  --primary-color: red;
}

p {
  color: var(--primary-color);
}

<!-- HTML -->
<p>I’m a red paragraph!</p>

/* JS */
var styles = getComputedStyle(document.documentElement);
var value = String(styles.getPropertyValue('--primary-color')).trim();
// value = 'red'
```

Similarly, to set the value of custom property at runtime, use the `setProperty()` method of the `CSSStyleDeclaration` object.

```
/* CSS */
:root {
  --primary-color: red;
}

p {
  color: var(--primary-color);
}

<!-- HTML -->
```

```
<p>Now I’m a green paragraph!</p>

/* JS */
document.documentElement.style.setProperty('--primary-color', 'green');
```

You can also set the value of the custom property to refer to another custom property at runtime by using the `var()` function in your call to `setProperty()`.

```
/* CSS */
:root {
  --primary-color: red;
  --secondary-color: blue;
}

<!-- HTML -->
<p>Sweet! I’m a blue paragraph!</p>

/* JS */
document.documentElement.style.setProperty('--primary-color', 'var(--secondary-co
```

Because custom properties can refer to other custom properties in your stylesheets, you could imagine how this could lead to all sorts of interesting runtime effects.

Browser support

Currently Chrome 49, Firefox 42, Safari 9.1, and iOS Safari 9.3 support custom properties.

Demo

Try out the sample for a glimpse at all of the interesting techniques you can now leverage thanks to custom properties.

Further reading

If you’re interested to learn more about custom properties, Philip Walton from the Google Analytics team has written a primer on why he’s excited for custom properties and you can keep tab on their progress in other browsers over on chromestatus.com.