Differences between CSS Custom Properties and Houdini Properties and values

CSS Custom Properties, also known as CSS Variables allow you to do awesome things. In this post we'll explore the different types of CSS Custom properties, what they are, how they work and which one to use in what circumstances.

The current version

The current version of CSS Custom Properties is a W3C Candidate Recommendation that define a way to set custom properties for our CSS content that we might want to dynamiclly change or that we want to reuse throughout the stylesheet.

What they are and how they work

The idea behind custom properties is to give developers the ability to create reusable properties and a way to use them. As explained in the Introduction to the specification:

> Large documents or applications (and even small ones) can contain quite a bit of CSS. Many of the values in the CSS file will be duplicate data; for example, a site may establish a color scheme and reuse three or four colors throughout the site. Altering this data can be difficult and error-prone, since it's scattered throughout the CSS file (and possibly across multiple files), and may not be amenable to Find-and-Replace.
>
> This module introduces a family of custom author-defined properties known collectively as custom properties, which allow an author to assign arbitrary values to a property with an author-chosen name, and the var() function, which allow an author to then use those values in other properties elsewhere

> in the document. This makes it easier to read large files, as seemingly-arbitrary values now have informative names, and makes editing such files much easier and less error-prone, as one only has to change the value once, in the custom property, and the change will propagate to all uses of that variable automatically.
>
> CSS Custom Properties for Cascading Variables Module Level 1 — Introduction

This sounds like a mouthfull so let's unpack it.

This API now allows developers to create custom properties to use in their stylesheets. It also introduces the `var` function to make use of these custom properties.

The example below defines two custom properties in the :root element for the stylesheet and later uses the propertie as the value of the var function.

```
:root {
  --main-color: #06c;
  --accent-color: #006;
}

/* The rest of the CSS file */
#foo h1 {
  color: var(--main-color);
}
```

This little example shows the basic syntax but imagine lager stylesheets where we use `--main-color` in multiple places. Now let's assume that marketing is changing the company colors. We only have to change the value of `--main-color` once and the changes will automatically change everywhere they are used.

If we want to override a specific instance we can just redeclare the custom property where we want to use a different value for the same variable

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

```
  --accent-color: #006;
}

/* The rest of the CSS file */
#foo h1 {
  color: var(--main-color);
}

#foo2 h1 {
  --main-color: rebeccapurple;
  color: var(--main-color);
}

#foo3 h1 {
  color: var(--main-color);
}
```

Houdini

Houdini is a joint effort by the W3C TAG and the W3C CSS Working Group to create APIs that will allow developers to tap into the internatls of the browser to get work done. Check Is Houdini Ready Yet? for status of the different APIs and their implementation across browsers.

The differences

As good as they are, custom properties, as defined in CSS have several significant drawbacks that Houdini addresses as explained in the sections below.

Inheritance

All custom properties will inherit down the cascade. There are times when you don't want this inherited behavior.

Houdinin custom properties let you choose whether the value inherits or not allowing for better encapsulation of styles.

Values

All custom properties defined in CSS are strings, regardless of what values we set them up. This makes them harder to work with in Javascript where we have to convert them to the actual value that we need.

Houdini props allow you to define specific value types for your properties. These are the same values used in other places in CSS so they will work the same throughout your stylesheet.

Animatable

Because they are strings, they don't animate or produce unexpected results.

Because Houdini properties use different values the browser can figure out how to animate those properties and if it's possible to animate them or not.

Validation

Because they are all considered to be strings, it's impossible to validate them with the proper value.

Houdini properties, on the other hand, have defined values that make validation possible and easy to use.

How they work

The first step in using custom properties is to define them in JavaScript. We use `CSS.registerProperty` to register the property with the CSS parser. It is always a good idea to check if the browser supports CSS.registerProperty before using it, this allows for fallbacks if it not.

```
if ('registerProperty' in CSS) {
  CSS.registerProperty({
    name: '--my-custom-prop',
    syntax: '<color>',
    inherits: true,
    initialValue: 'black'
  });
} else {
```

```
  console.log('registerProperty is not supported');
}
```

There are 4 values that we need to pass to `registerProperty`

Name
: The name is what we'll use to reference the property. The two dashes at the beginning should be familiar from CSS variables and are required. This is how we'll distinguish our custom variables and properties from what the CSS WG does and will do in the future.

Syntax
: indicates the possible syntaxes for the property. The following values are available in level 1 of the spec and matching corresponding units in CSS Values and Units Module Level 3 (check that specification for a full list and the [CSS Properties and Values API Level 1]())

  You can create fairly complex syntax for your custom properties but until we become familiar with them, I advocate for the KISS (Keep It Simple Silly) principle.

inherits
: Inherits tells the CSS parser if this custom rule should propagate down the cascade. Setting it to false gives us more power to style specific elements without being afraid to mess up elements further down the chain.

initialValue
: Use this to provide a sensible default for the property. We'll analyze why this is important later.

  That's it... we now have a custom property.

Using custom properties

To demonstrate how to use Custom Properties we'll reuse the --bg-color Javascript example and use it in several different elements.

```
CSS.registerProperty({
  name: '--bg-color',
```

```
  syntax: '<color>',
  inherits: false,
  initialValue: 'red'
});
```

The CSS will not be any different than if we used variables. But the things it does for free are much more interesting.

First we define common parameters to create 200px by 200px squares using div elements.

The examples below use SCSS syntax to make the code easier to read; It is also important to note that SCSS variables are not the same as CSS or Houdini custom properties so having both in the same stylesheet will not cause any problems.

```
div {
  border: 1px solid black;
  height: 200px;
  width: 200px;
}
```

`.smoosh1` and `.smoosh2` set up colors other than the initial value and each has a different color to change on hover.

```
.smoosh1 {
  --bg-color: rebeccapurple;
  background: var(--bg-color);
  transition: --bg-color 0.3s linear;
  position: absolute;
  top: 50vh;
  left: 15vw;

  &:hover {
    --bg-color: orange;
  }
}
```

```
.smoosh2 {
  --bg-color: teal;
  background: var(--bg-color);
  transition: --bg-color 0.3s linear;
  position: absolute;
  top: 20em;
  left: 45em;

  &:hover {
    --bg-color: pink;
  }
}
```

`.smoosh3` was set up with a wrong type of color (1 is not a valid CSS color). In normal CSS the rule would be ignored and there would be no background color. Because we added an initialValue to the property, it'll take this value instead of giving an error or needing a fallback.

```
.smoosh3 {
  --bg-color: 1;
  background: var(--bg-color);
  transition: --bg-color 0.3s linear;
  position: absolute;
  top: 5em;
  left: 35em;

  &:hover {
    --bg-color: lightgrey;
  }
}
```

When would you use which?

This is a tricky question. Most of the time you'll want to use the Houdini version that allows you tighter control over the individual properties.

But We have to consider that only recent versions of some modern browsers (partial support in Safari TP and Chrome and under development for Firefox) support the Javascript API.

So, in the end, it depends on what you need. Most of the time your production code should use the CSS-only version of custom properties or use Houdini properties with a CSS-only fallback like the code below. This way, if the browser doesn't support Houdini we still have a custoom property that the CSS code can use.

```
if ('registerProperty' in CSS) {
  CSS.registerProperty({
    name: '--my-custom-prop',
    syntax: '<color>',
    inherits: true,
    initialValue: 'black'
  });
} else {
  console.log('registerProperty is not supported');
  console.log('reverting to old-style properties')
  const sheet = document.styleSheets[0];
  sheet.insertRule(":root { --my-custom-prop: #000000 }", 1);
}
```