

# Towards a Color API for the Web Platform

**Lea Verou, Chris Lilley**

# User/Platform Needs

---

- Input and output for APIs (Canvas, CSS OM, WebGPU, Eyedropper API, `<input type=color>` etc)
- Parsing CSS colors
- Conversion between color spaces (gamut mapping optional)
- Color manipulation
- Color difference
- Contrast
- Interpolation

# Requirements

---

- Color space agnostic
- HDR compatible
- Extensible
- Layered design:
  - usable by non-experts
  - useful to experts

# CSSColorValue

---

- Draft:  
<https://drafts.css-houdini.org/css-typed-om/#colorvalue-objects>
- Part of Typed OM
- Abstract CSSColorValue inherits from CSSStyleValue
- Multiple subclasses: CSSRGB, CSSHSL, CSSColor etc with different API shapes
  - CSSRGB has `.r`, `.g`, `.b`, CSSColor has `.channels`
  - `new CSSRGB(0, 1, 0)` but `new CSSColor("srgb", [0, 1, 0])`
- Color space conversion: `color.to("hsl")`
- Early stage spec, no implementations

# Advantages of CSSColorValue

---

- Only one object across the Web Platform
- Good integration with CSS out of the box

# Problems with CSSColorValue

---

- **Designed to represent CSS <color> values, not colors**
- Two representations for sRGB colors: CSSRGB and CSSColor with `type="srgb"`
  - `color.to("rgb")` returns an CSSRGB object
  - `color.to("srgb")` returns a CSSColor object
- Keywords are not strings, but CSSKeywordValue
  - Strings are accepted for input, but output is objects
  - E.g. `color.colorSpace.value` to read the color space id

# Problems with CSSColorValue (cont'd)

---

- Coordinates are not numbers, but CSSNumberish
  - Numbers are accepted for input, but output is objects
  - E.g. `color.r.value` to read the red coordinate
  - Coordinates may even correspond to `calc()` expressions and the like
- These issues cannot be rectified by API design iteration
- They are inherent to the fact that `CSSColorValue` is designed to represent **syntactic constructs**

# Color Object: History

- In 2020, Chris and I started Color.js (colorjs.io), to play around with API design ideas and algorithms
- A fair bit of community input and usage, despite no "official" release



*Let's get serious about color*

- ✓ **Fully color space aware**  
Each color belongs to a color space; operations are color space agnostic. Modules for [a wide variety of color spaces](#), including Lab/LCH, sRGB and friends (HSL/HSV/HWB), Display P3, J<sub>2</sub>a<sub>2</sub>b<sub>2</sub>, REC.2100 and more.
- ✓ **Up to date with CSS Color 4**  
Every [CSS Color 4](#) format & color space supported for both [input](#) and [output](#), whether your browser supports it or not.
- ✓ **Modular & Extensible**  
Use only what you need, or a bundle. Client-side or Node. Deep extensibility with [hooks](#).
- ✓ **Doesn't gloss over color science**  
Actual [gamut mapping](#) instead of naïve clipping, multiple [DeltaE](#) methods (76, CMC, 2000, J<sub>2</sub>), multiple [chromatic adaptation](#) methods (von Kries, Bradford, CAT02, CAT16), all with sensible defaults
- ✓ **Readable, object-oriented API**  
Color objects for multiple operations on the same color, and static [Color.something\(\)](#) functions for one-off calculations



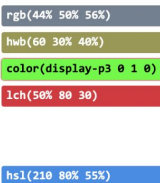
**WARNING:** Color.js is currently an unreleased work in progress. Here be dragons. If you found this website somehow, feel free to try it out and [give us feedback](#), but sshhhh! 😬 There are more bugs in the live code snippets ("Color Notebook") in the docs than the actual Color.js library, so before reporting a bug, please try to reproduce it outside Color Notebook.

## Reading colors

Any color from CSS Color Level 4 should work:

```
let color = new Color("slategray");
let color2 = new Color("hwb(60 30% 40%)");
let color3 = new Color("color(display-p3 0 1 0)");
let color4 = new Color("lch(50% 80 30)");

// CSS variables work too!
let colorjsBlue = new Color("--color-blue");
```





# Color Object

---

- Draft spec: <https://projects.verou.me/color-api/>
- Draft explainer: <https://github.com/LeaVerou/color-api>
- While our experience with color.js has influenced the design, this API has been designed *from scratch* with the different needs of a built-in API in mind and has a quite different shape





# Current API Draft

---

- One Color class
- Constructors:
  - `new Color(colorSpace, coords [,alpha])`
  - `new Color({colorSpace, coords, alpha})`
  - `new Color(color) // string, CSSColorValue, or other Color instance`
- `color.coords` is an `ObservableArray`
- `color.colorSpace` is a string or `ColorSpace` object
- `color.alpha` is a number
- All currently mutable

# Current API Draft: Color spaces

---

- Color spaces are represented by **ColorSpace** objects
- They may be registered via **ColorSpace.register(id, options)** or anonymous ( of   )
- Registered color spaces can be referred to by id
- Color spaces declare their white point, coordinates, gamut, and conversion code to/from any existing space
- **ColorSpace.load(iccProfile)** returns a **Promise<ColorSpace>**
- Color spaces cannot become un-registered

# Current API: Color conversion & manipulation

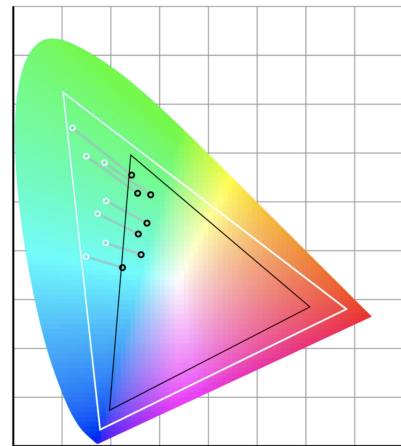
---

- Read
  - `color.get("lch", "l")` to get any coordinate, in any space
  - `color.get("l")` for current color space
  - `color.to("lab")` // new Color instance
- Mutable: **`color.set("lch", "l", 80)`**
- Relative manipulations:
  - `color.set("lch", "l", color.get("lch", "l") * .8)`
  - `color.set("lch", "l", l => l * .8)`
- Aggregate:
  - `color.set("lch", {l: l => l * 1.2, c: c => c + 20, h: 10})`

# Current API: Gamut mapping

---

- Everything is lossless by default
- **`color.inGamut(optional space)`** to check if the color is in gamut of its own or other color spaces
- Explicit gamut mapping via **`color.toGamut(optional space, optional method)`**
  - LCH chroma reduction by default, but it can use any coordinate of any color space (though some would be nonsensical, e.g. hue)



# Open problems

---

- How to declare polar color spaces? ([#6](#))
- What is a color space? ([w3c/css-houdini-drafts#1044](#))
  - hsl, hwb, srgb are all representations of the same color space, sRGB
  - Same with lab and lch
  - Should that be declared syntactically?
  - Should there be a way to read the actual color space?
- Mutable or immutable? Currently a mix:
  - **color.set()** mutates in place, **color.coords** is mutable
  - **color.to()** and **color.toGamut()** create a new instance

# Open problems

---

- HDR tone mapping?
- Interaction between the Color API and CSS
  - Are registered color spaces available in CSS too?
  - Do color spaces declared via **@color-profile** become registered ColorSpace objects once the profile loads?