Towards a Color API for the Web Platform

Lea Verou, Chris Lilley

Hi, I'm Lea!

- Elected W3C TAG member
- W3C CSS WG member
- Co-editor of CSS Color 4, CSS Color 5
- HCI researcher at MIT CSAIL
- lea@verou.me
- Blog: lea.verou.me

User/Platform Needs

- Input and output for Web APIs (Canvas, CSS OM, WebGPU, Eyedropper API, <input type=color> etc)
- Parsing CSS colors
- Conversion between color spaces (with or without gamut mapping)
- 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 colorSpace="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, usage, even derivative work, despite no "official" release

Color API

- Draft spec: wicg.github.io/color-api
- Draft explainer: github.com/wicg/color-api
- API designed from scratch
- Color.js is a strong influence, but API shape is different as this is a native API, not a library
- Useful feedback from Tab Atkins has improved the API

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

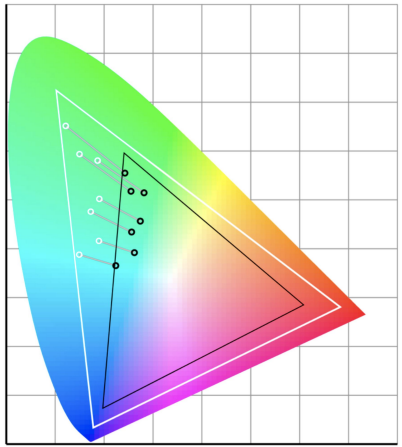
- Color spaces are represented by **ColorSpace** objects
- They may be registered via **ColorSpace.register(id, options)** or stay anonymous (🥫 of 🪱🪱🪱)
- Registered color spaces can be referred to by id
- Color spaces declare their white point, coordinates, gamut, and conversion code to/from any existing 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

- Conversions are lossless by default
- **color.inGamut(optional space)** to check if in gamut (of own or other color spaces)
- Explicit gamut mapping via **color.toGamut(optional space, optional method)**
  - LCH chroma reduction by default, configurable
  - Open problem how to avoid gamut mapping via nonsensical coordinates (e.g. hue)



CSS WG breakout on July 21st 2021

- Discussed these options to resolve on direction
- Minutes: w3c/css-houdini-drafts#1047
- **RESOLVED:** Add Color API for representing color points, separate from CSSColorValue to represent CSS color values
- **RESOLVED:** Color API should handle all the color spaces currently specced for the web platform.
- Color API repo moved to WICG for incubation

Open issues

- 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 issues

- HDR tone mapping?
- Integration between 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?
- How does parsing and serialization of author-defined color spaces work?

Interested? Come help us design this!

github.com/wicg/color-api