Spectrum library documentation

Table of contents

What is Spectrum?

Welcome to Spectrum, a lightweight JavaScript / TypeScript library designed to simplify color manipulation and conversion tasks within the `RGB`, `HSL`, and `HEX` color spaces.

It may be not the most extensive library out there, but it's precisely what you need for common color-related tasks. Whether you want to blend two colors, get a darker version of your color, or the saturation of a HEX color value. Spectrum is your finely-tuned instrument for simplifying these processes.

Key Features

- **Color values conversion**: convert color values within HEX, HSL, and RGB color spaces.
- **Effortless color mixing**: combine two colors with ease.
- **Color paramenters adjustments**: lightness, saturation, opacity, and more.
- **Color Inversion**: get a negative color of the given one.
- **Color Palette Generation:** create a color palette with varying lightness levels, all derived from a single base color.

Spectrum is here to make your color-related tasks more efficient and straightforward. In this documentation, we'll explore how to leverage its capabilities. Let's dive in!

Getting started

Spectrum is a lightweight library with no dependencies, making it compatible with any JavaScript environment. This page will guide you through the installation process and provide a simple example of how to use Spectrum for color manipulation.

Installation

To start using Spectrum in your project, simply run one of the following commands in your terminal:

npm | Bun | Deno | pnpm | Yarn

```
npx jsr add @particles/spectrum

# or via ordinary npm install
npm i @snipshot/spectrum
```

Example usage

```
import Spectrum, { adjustHsl } from '@snipshot/spectrum';

// Create a new Spectrum instance in the HSL color space
const spectrum = new Spectrum('hsl', [231, 0.66, 0.53, 0.8]);

// Adjust the hue and lightness values of the color
const adjustedColor = adjustHsl(spectrum, { hue: -23, lightness: '-13%' });

console.log(adjustedColor.hsl); // { h: 208, s: 0.66, l: 0.4, a: 0.8 }
console.log(adjustedColor.hex); // #236aa9cc
```

In the example above, we initiate a new `Spectrum` instance using one of the available color spaces: `hex`, `hsl`, or `rgb` as the first argument. The second argument can be an

`array` or a `string`. For more details, refer to the Spectrum class API reference.

After initializing the instance, you can apply transformation methods provided by the library. In this example, we change `hue` and `lightness` values, resulting in a new Spectrum instance with the modified color. You can then access the hsl and hex properties of the new color.

Next steps

For more in-depth information about using Spectrum, explore the API Documentation section.

Thank you for choosing **Spectrum**! 😉

Spectrum class

The `Spectrum` class is a fundamental component of the library, representing colors in `HEX`, `HSL`, and `RGB` color spaces. Spectrum instances can be used with various methods to convert between color spaces and access individual color channels.

```
// Create a Spectrum instance from a hex color value
const spectrum = new Spectrum('hex', '#FF0000');

spectrum.rgb; // { r: 255, g: 0, b: 0, a: 1 }
spectrum.hsl; // { h: 0, s: 1, l: 0.5, a: 1 }
spectrum.hex; // "#ff0000"
```

Constructor

```
new Spectrum('colorSpace', value);
```

Parameters

Name	Type	Description
`colorSpace`	`'hex' | 'hsl' | 'rgb' | CssNamedColor`	The color space of the input value
`value`	`string | Array<string | number> | undefined`	The color value. The format depends on the color space. See details.

Value

The allowed input formats for each color space are as follows:

For `hex` color space:

- The input can be only a string value with optional preceding `#`.
- It also accepts shorthand HEX notation and alpha channel. See examples.

For `hsl` and `rgb` color spaces an input can be:

- A string of space-separated or comma-separated values.
- An array of values in a valid format.

For `hsl`, the format is `[hue, saturation, lightness, opacity]`.

For `rgb`, the format is `[red, green, blue, opacity]`.

You can also use a CSS named color as a first parameter. For example, `'red'` or `'blue'`. In this case, you should not provide a second parameter. See examples.

Valid formats

For `hsl`:

Value	Format	Example
hue	`string | number` without a unit	`180`, `'180'`
saturation	Percentage `string` or a decimal point `number` in range `[0; 1]`	`'25%'`, `0.25`
lightness	Percentage `string` or a decimal point `number` in range `[0; 1]`	`'50%'`, `0.5`
opacity	Percentage `string` or a decimal point `number` in range `[0; 1]`	`'90%'`, `0.9`

For `rgb`:

Value	Format	Example
red	`string | number`	`255`, `'255'`
green	`string | number`	`'90'`, `90`

Value	Format	Example
blue	`string | number`	`'30'`, `30`
opacity	Percentage `string` or a decimal point `number` in range `[0; 1]`	`'90%'`, `0.9`

Examples

With hex

```
new Spectrum('hex', '#AE2127');
new Spectrum('hex', 'ae2127');
new Spectrum('hex', '236aa9cc');

new Spectrum('hex', '#eee');
new Spectrum('hex', 'EEE');
new Spectrum('hex', '#ea3c');
```

With hsl

```
new Spectrum('hsl', '180 0.3 0.9');
new Spectrum('hsl', '180, 0.3, 0.9');
new Spectrum('hsl', '180, 0.3, 0.9, 20%');
new Spectrum('hsl', '180 30% 90% 0.2');

new Spectrum('hsl', [180, 0.3, 0.9]);
new Spectrum('hsl', [180, 0.3, 0.9, 0.2]);
new Spectrum('hsl', [180, '30%', '90%', '20%']);
new Spectrum('hsl', [180, '30%', '90%', 0.2]);
```

With rgb

```
new Spectrum('rgb', '255 255 255');
new Spectrum('rgb', '255, 255, 255');
new Spectrum('rgb', '255, 255, 255, 20%');
new Spectrum('rgb', '255 255 255 0.2');

new Spectrum('rgb', [255, 255, 255]);
new Spectrum('rgb', ['255', '255', '255']);
new Spectrum('rgb', [255, 255, 255, 0.2]);
new Spectrum('rgb', ['255', '255', '255', '20%']);
new Spectrum('rgb', ['255', '255', '255', '0.2']);
```

With CSS named color

```
new Spectrum('red');
new Spectrum('blue');
new Spectrum('lightseagreen');
```

Static class methods

Apart from the constructor, you can also create a new `Spectrum` instance: using the class methods `fromHslObj` and `fromRgbObj`. These methods allow you to create a new instance from the objects returned by `hsl` and `rgb` instance properties or by providing a custom object of your own.

:::caution The objects passed to `fromHslObj()` and `fromRgbObj()` methods must be numeric values. Thus, valid value for the saturation property is only `s: 0.7`. Setting it as `s: '70%'` will result in an error. :::

fromHslObj()

Creates a new instance of the `Spectrum` class using an HSL object as an input. All properties of an HSL object are required.

Usage

```
Spectrum.fromHslObj({ h: 8, s: 0.5, l: 0.41, a: 0.9 }: HslObj);
```

Parameters

Name	Type	Description
`hslObj`	`HslObj`	An object representing the HSL color values with properties `h` (hue), `s` (saturation), `l` (lightness), and `a` (alpha).

Returns `Spectrum` instance.

Examples

```
const color = Spectrum.fromHslObj({ h: 180, s: 0.5, l: 0.75, a: 1 });
console.log(color.hsl); // { h: 180, s: 0.5, l: 0.75, a: 1 }

const green = new Spectrum('rgb', [0, 255, 0]);
const greenCopy = Spectrum.fromHslObj(green.hsl);
```

---

fromRgbObj()

Creates a new instance of the `Spectrum` class using an RGB object as an input. All properties of an RGB object are required.

Usage

```
Spectrum.fromRgbObj({ r: 255, g: 0, b: 0, a: 1 }: RgbObj);
```

Parameters

Name	Type	Description
`rgbObj`	`RgbObj`	An object representing the RGB color values with properties `r` (red), `g` (green), `b` (blue), and `a` (alpha).

Returns `Spectrum` instance.

Examples

```
const color = Spectrum.fromRgbObj({ r: 255, g: 130, b: 60, a: 0.8 });
console.log(color.rgb); // { r: 255, g: 130, b: 60, a: 0.8 }

const red = new Spectrum('rgb', [255, 0, 0]);
const redCopy = Spectrum.fromRgbObj(red.rgb);
```

Instance properties

hex

The `hex` property retrieves the hexadecimal representation of the color.

Returns: `string`.

```
const color = new Spectrum('hex', '412ED1');
color.hex; // #412ed1
```

---

hsl

The `hsl` property retrieves the HSL object of the color.

Returns `HslObj`.

```
const color = new Spectrum('hsl', '180 70% 50% 82%');
color.hsl; // { h: 180, s: 0.7, l: 0.5, a: 0.82 }
```

rgb

The `rgb` property retrieves the RGB object of the color.

Returns `RgbObj`.

```
const color = new Spectrum('rgb', '230 90 115 82%');
color.rgb; // { r: 230, g: 90, b: 115, a: 0.82 }
```

alpha

The `alpha` property retrieves the alpha channel value of the color.

Returns: `number`.

```
const color = new Spectrum('rgb', '230 90 115 82%');
color.alpha; // 0.82
```

red

The `red` property retrieves the red channel value of the color.

Returns: `number`.

```
const color = new Spectrum('rgb', '230 90 115 82%');
color.red; // 230
```

green

The `green` property retrieves the green channel value of the color.

Returns: `number`.

```
const color = new Spectrum('rgb', '230 90 115 82%');
color.green; // 90
```

blue

The `blue` property retrieves the blue channel value of the color.

Returns: `number`.

```
const color = new Spectrum('rgb', '230 90 115 82%');
color.blue; // 115
```

hue

The `hue` property retrieves the hue value of the color.

Returns: `number`.

```
const color = new Spectrum('hsl', '180 70% 50% 82%');
color.hue; // 180
```

saturation

The `saturation` property retrieves the saturation value of the color.

Returns: `number`.

```
const color = new Spectrum('hsl', '180 70% 50% 82%');
color.saturation; // 0.7
```

lightness

The `lightness` property retrieves the lightness value of the color.

Returns: `number`.

```
const color = new Spectrum('hsl', '180 70% 50% 82%');
color.lightness; // 0.5
```