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
```

adjustHsl()

The `adjustHsl` function allows you to adjust the HSL (Hue, Saturation, Lightness) values of a color object. This function returns a new `Spectrum` instance with the updated HSL values.

Usage

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

const color = new Spectrum('hsl', [200, 0.5, 0.6, 1]);

const adjustedColor = adjustHsl(color, {
  hue: -20, // Adjust hue by -20 degrees
  saturation: 0.1, // Increase saturation by 10%
  lightness: -0.05 // Decrease lightness by 5%
});

console.log(adjustedColor.hsl); // { h: 180, s: 0.6, l: 0.55, a: 1 }
console.log(color.hex === adjustedColor.hex); // false
```

Parameters

`adjustHsl(colorObj, options)`

Parameter	Type	Required	Valid range	Description
`colorObj`	Spectrum instance	true	-	The Spectrum instance representing the color you want to adjust

Parameter	Type	Required	Valid range	Description
`options.hue`	`number`	false	`[-360; 360]`	The amount by which to adjust the hue value
`options.saturation`	`string`	false	`['-100%'; '100%']`	The amount by which to adjust the saturation value. Should be provided as a percentage string.
`options.lightness`	`string`	false	`['-100%'; '100%']`	The amount by which to adjust the lightness value. Should be provided as a percentage string.
`options.alpha`	`number`	false	`[-1; 1]`	The amount by which to adjust the alpha value

Return Value

The `adjustHsl` function returns a new `Spectrum` instance with the adjusted HSL values.

Examples

Adjust all properties

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

const color = new Spectrum('hsl', [200, 0.5, 0.6, 1]);
const adjustedColor = adjustHsl(color, {
  hue: -45,
  saturation: 0.3,
  lightness: -0.14,
  alpha: -0.32
});

console.log(adjustedColor.hsl); // { h: 155, s: 0.2, l: 0.46, a: 0.68 }
```

Adjust Hue

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

const color = new Spectrum('hsl', [200, 0.5, 0.6, 1]);
const adjustedColor = adjustHsl(color, { hue: -20 });

console.log(adjustedColor.hsl); // { h: 180, s: 0.5, l: 0.6, a: 1 }
```

adjustRgb()

The `adjustHsl` function allows you to adjust the RGB (Red, Green, Blue) values of a color object. This function returns a new `Spectrum` instance with the updated RGB values.

Usage

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

const color = new Spectrum('rgb', [255, 0, 0, 1]);

const adjustedColor = adjustRgb(color, {
  red: -50, // Adjust red by -50
  alpha: -0.5 // Adjust alpha by -0.5
});

console.log(adjustedColor.rgb); // { r: 205, g: 0, b: 0, a: 0.5 }
console.log(color.hex === adjustedColor.hex); // false
```

Parameters

`adjustRgb(colorObj, options)`

Parameter	Type	Required	Valid range	Description
`colorObj`	Spectrum instance	true	-	The Spectrum instance representing the color you want to adjust
`options.red`	`number`	false	`[-255; 255]`	The amount by which to adjust the red channel value

Parameter	Type	Required	Valid range	Description
`options.green`	`number`	false	`[-255; 255]`	The amount by which to adjust the green channel value
`options.blue`	`number`	false	`[-255; 255]`	The amount by which to adjust the blue channel value
`options.alpha`	`number`	false	`[-1; 1]`	The amount by which to adjust the alpha channel value

Return Value

The `adjustRgb` function returns a new `Spectrum` instance with the adjusted RGB values.

Examples

Adjust all properties

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

const color = new Spectrum('hsl', [108, 90, 50, 0.5]);
const adjustedColor = adjustHsl(color, {
  red: 90,
  green: -25,
  blue: 102,
  alpha: 0.32
});

console.log(adjustedColor.rgb); // { r: 198, g: 65, b: 152, a: 0.82 }
```

Adjust Hue

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

const color = new Spectrum('rgb', [255, 190, 0, 1]);
const adjustedColor = adjustRgb(color, { green: -28 });

console.log(adjustedColor.rgb); // { r: 255, g: 162, b: 0, a: 1 }
```

colorMix()

The `colorMix` function allows you to mix two colors according to a specified weight in RGB color space. This function returns a new `Spectrum` instance representing the resulting color.

Usage

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

const red = new Spectrum('hex', '#f00');
const blue = new Spectrum('rgb', '0, 0, 255, 1');

const purple = colorMix(red, blue, 0.5); // 0.5 is a weight of the first color (max value is 1)

console.log(purple.rgb); // { r: 128, g: 0, b: 128, a: 1 }
```

Parameters

`colorMix(color1, color2, weight)`

Parameter	Type	Required	Valid range	Description
`color1`	Spectrum instance	true	-	The first color to mix
`color2`	Spectrum instance	true	-	The second color to mix
`weight`	`number`	true	`[0; 1]`	The weight of the **first color** in the mixture.

> **Note**
>
> The `weight` parameter determines the influence of the first color (`color1`) on the resulting mixed color. The closer the `weight` value is to 1, the more dominant `color1` will be in the mixture. Thus, when the `weight` is set to 0.5, both colors, `color1` and `color2`, are given equal importance in the blend.

Return Value

The `colorMix` function returns a new `Spectrum` instance representing the mixed color.

Examples

Mix two hex colors

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

const darkSlateGrey = new Spectrum('hex', '#2F4F4F');
const orange = new Spectrum('hex', '#FFA500');

const mix = colorMix(darkSlateGrey, orange, 0.3); // 30% of Dark slate grey and
70% of Orange;

console.log(mix.hsl); // { h: 41, s: 0.77, l: 0.43, a: 1 }
```

Mix colors with opacity

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

const fuchsia = new Spectrum('rgb', '255 0 255 0.3');
const midnightBlue = new Spectrum('rgb', '25 25 112 0.65');

const mix = colorMix(fuchsia, midnightBlue, 0.5); // 50% of Fuchsia and 50% of
Midnight blue;

console.log(mix.rgb); // { r: 140, g: 13, b: 184, a: 0.47 }
```

createPalette()

The `createPalette` function allows you to generate a palette of colors based on a given `Spectrum` instance. Each key in the palette object represents a lightness value from 0 to 100, and the corresponding value is a `Spectrum` object.

Usage

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

const cyan = new Spectrum('hex', '#0ff');
console.log(cyan.hsl); // { h: 180, s: 1, l: 0.5, a: 1 }

const palette = createPalette(cyan);

console.log(palette[0].hsl); // { h: 180, s: 1, l: 0, a: 1 } - black
console.log(palette[44].hsl); // { h: 180, s: 1, l: 0.44, a: 1 }
console.log(palette[70].hsl); // { h: 180, s: 1, l: 0.7, a: 1 }
console.log(palette[100].hsl); // { h: 180, s: 1, l: 1, a: 1 } - white
```

Parameters

`createPalette(colorObj)`

Parameter	Type	Required	Description
`colorObj`	Spectrum instance	true	The color from which will be generated a palette

Return Value

The `colorMix` function returns a palette object with lightness values from 0 to 100 as keys, where each key corresponds to a `Spectrum` instance representing a color with the specified lightness. The keys step is equal to 1, thus, there are 101 values inside the palette object.

getSplitComplementary()

The `getSplitComplementary` function allows you to generate split complementary colors based on a given `Spectrum` instance. If you are not familiar with the term, you may find useful this article: "What Are Split-Complementary Colors? Best Ways to Use This Color Scheme".

Usage

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

const cyan = new Spectrum('hsl', [180, 1, 0.5]);

const { secondary, tertiary } = getSplitComplementary(cyan);

console.log(secondary.hsl); // { h: 330, s: 1, l: 0.5, a: 1 } – rose
console.log(tertiary.hsl); // { h: 30, s: 1, l: 0.5, a: 1 } – dark orange
```

Parameters

`getSplitComplementary(colorObj)`

Parameter	Type	Required	Description
`colorObj`	`Spectrum` instance	true	The base color for complementary colors generation

Return Value

The `getSplitComplementary` function returns a an object with two keys: `secondary` and `tertiary`, the generated split complementary colors for the given one. The values are instances of `Spectrum` class.

```
type Return = {
  secondary: Spectrum;
  tertiary: Spectrum;
};
```

getTriadic()

The `getTriadic` function allows you to generate triadic colors combination based on a given `Spectrum` instance. If you are not familiar with the term, you may find useful this article: "What Are Triadic Colors and How Are They Used? Triadic Color Schemes Explained".

Usage

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

const cyan = new Spectrum('hsl', [180, 1, 0.5]);

const [secondary, tertiary] = getTriadic(cyan);

console.log(secondary.hsl); // { h: 300, s: 1, l: 0.5, a: 1 } - magenta
console.log(tertiary.hsl); // { h: 60, s: 1, l: 0.5, a: 1 } - yellow
```

Parameters

`getTriadic(colorObj)`

Parameter	Type	Required	Description
`colorObj`	`Spectrum` instance	true	The base color for triadic colors generation

Return Value

The `getTriadic` function returns a an `Array` with two `Spectrum` instances, that are triadic colors for the given one.

```
type Return = [Spectrum, Spectrum];
```

hexToRgb()

The `hexToRgb` function allows you to convert a hexadecimal color value to its corresponding RGB value.

Usage

```
import { hexToRgb } from '@snipshot/spectrum';

const rgbObj = hexToRgb('#ff0000');

console.log(rgbObj); // { r: 255, g: 0, b: 0, a: 1 }
```

Parameters

`hexToRgb(colorValue)`

Parameter	Type	Required	Description
`colorValue`	`string`	true	HEX code of the color with optional #. Accepts shorthand notation and alpha channel.

Return Value

The `hexToRgb` function returns an RGB object of the color value.

Examples

Shorthand notation

```
import { hexToRgb } from '@snipshot/spectrum';

const lightBlueRgb = hexToRgb('#3ae');
console.log(lightBlueRgb); // { r: 51, g: 170, b: 238, a: 1 }

const green = hexToRgb('1e3');
console.log(green); // { r: 17, g: 238, b: 51, a: 1 }
```

Colors with opacity

```
import { hexToRgb } from '@snipshot/spectrum';

const lily = hexToRgb('#7777eeb3');
console.log(lily); // { r: 119, g: 119, b: 238, a: 0.7 }

const darkOrange = hexToRgb('#941a');
console.log(darkOrange); // { r: 153, g: 68, b: 17, a: 0.67 }
```

hslToRgb()

The `hslToRgb` function allows you to convert an HSL color value to its corresponding RGB value.

Usage

```
import { hslToRgb } from '@snipshot/spectrum';

const salmon = hslToRgb({ h: 6, s: 0.93, l: 0.71, a: 1 });

console.log(salmon); // { r: 250, g: 126, b: 112, a: 1 }
```

Parameters

`hslToRgb(hslObj)`

Parameter	Type	Required	Description
`hslObj`	HslObj	true	An HSL object that represents a color in the HSLA (Hue, Saturation, Lightness, Alpha) color space.

Return Value

The `hslToRgb` function returns an RGB object of the color value.

Examples

With opacity

```
import { hexToRgb } from '@snipshot/spectrum';

const teal = hslToRgb({ h: 180, s: 1, l: 0.25, a: 0.75 });
console.log(teal); // { r: 0, g: 128, b: 128, a: 0.75 }

const violet = hslToRgb({ h: 300, s: 0.76, l: 0.72, a: 0.32 });
console.log(violet); // { r: 238, g: 129, b: 238, a: 0.32 }
```

invert()

The `invert` function allows you to get a negative color. This function returns a new `Spectrum` instance representing the inverted color.

Usage

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

const yellow = new Spectrum('rgb', [255, 255, 0]);
const negativeColor = invert(yellow, 1); // 1 is a weight of the inverted color

console.log(negativeColor.rgb); // { r: 0, g: 0, b: 255, a: 1 } - blue
```

Parameters

`invert(color1Obj, weight)`

Parameter	Type	Required	Valid range	Description
`colorObj`	Spectrum instance	true	-	The initial color
`weight`	`number`	true	`[0; 1]`	The weight of the **inverted color** in the result

> **Note**
>
> Similar to the behaviour in the `colorMix` function, the weight parameter controls the influence of the inverted color on the resulting color. The closer the `weight` value is to 1, the more dominant the inverted color will be.
> Thus, when the `weight` is set to 0, the resulting color remains identical to the initial colorObj color.

Return Value

The `invert` function returns a new `Spectrum` instance representing the negative color.

Examples

Weight is 0.5

When the `weight` value is equal to 0.5 (50% of initial color and 50% of inverted color), it will always produce grey color (`#808080`).

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

const yellow = new Spectrum('rgb', [255, 255, 0]);
const negativeYellow = invert(yellow, 0.5);

console.log(negativeYellow.hex); // #808080

const crimson = new Spectrum('hex', '#DC143C');
const negativeCrimson = invert(crimson, 0.5);

console.log(negativeCrimson.hex); // #808080
```

Weight is 0

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

const yellow = new Spectrum('rgb', [255, 255, 0]);
const negativeColor = invert(yellow, 0);

console.log(negativeColor.rgb); // { r: 255, g: 255, b: 0, a: 1 } - initial
color
```

onBgColor()

The `onBgColor` helps you choose a proper color considering your background. You should provide a `Spectrum` instance of your background color as a first argument, and an object with two color options as a second.

> **Note**
>
> `dark` property value of the options objects specifies the dark color that may be used on the **light** background. Otherwise, `light` property value specifies the light color that may be used on the **dark** background.

Usage

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

const darkBlueBackground = new Spectrum('hex', '#00008B');

const onDarkBlueBackground = onBgColor(darkBlueBackground, {
  dark: new Spectrum('hex', '#000'), // black, can be used on light backgrounds
  light: new Spectrum('hex', '#fff') // white, can be used on light backgrounds
});

console.log(onDarkBlueBackground.hex); // #ffffff - white
```

Parameters

`onBgColor(colorObj, options)`

Parameter	Type	Required	Description
`colorObj`	Spectrum instance	true	The background color

Parameter	Type	Required	Description
`options.dark`	`string | Spectrum`	true	The color to use on the **light** background
`options.light`	`string | Spectrum`	true	The color to use on the **dark** background

Return Value

`string | Spectrum`

The `onBgColor` one of the values provided inside the `options` object.

Examples

Light gray background

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

const lightGrayBg = new Spectrum('rgb', [200, 200, 200]);
const options = {
  dark: '#000000',
  light: '#ffffff'
};

const onColor = onBgColor(lightGrayBg, options);
console.log(onColor); // '#000000'
```

Medium slate blue background

```
const mediumSlateBlue = new Spectrum('hex', '7B68EE');
const options = {
  dark: new Spectrum('hex', '#111'),
  light: new Spectrum('hex', '#eee')
};

const onColor = onBgColor(colorObj, options);
console.log(onColor.hex); // '#eeeeee'
```

rgbObjToHex()

The `rgbObjToHex` function allows you to convert an RGB object to its corresponding hexadecimal color code.

Usage

```
import { rgbObjToHex } from '@snipshot/spectrum';

const hex = rgbObjToHex({ r: 255, g: 255, b: 0, a: 1 });

console.log(hex); // #ffff00
```

Parameters

`rgbObjToHex(rgbObj)`

Parameter	Type	Required	Description
`rgbObj`	RgbObj	true	An RGB object that represents a color in the RGBA (Red, Green, Blue, Alpha) color space.

Return Value

The `rgbObjToHex` function returns the hexadecimal value of the color as a `string`. If the input object's alpha channel value is not equal to 1, the resulting hex code will include an additional number representing the alpha channel.

Examples

A color with opacity

```
import { rgbObjToHex } from '@snipshot/spectrum';

const hex = rgbObjToHex({ r: 138, g: 217, b: 16, a: 0.36 });

console.log(hex); // #8ad9105c
```

rgbObjToHsl()

The `rgbObjToHsl` function allows you to convert an RGB object to its corresponding HSLA (Hue, Saturation, Lightness, Alpha) values.

Usage

```
import { rgbObjToHsl } from '@snipshot/spectrum';

const hsl = rgbObjToHsl({ r: 255, g: 255, b: 0, a: 1 });

console.log(hsl); // { h: 60, s: 1, l: 0.5, a: 1}
```

Parameters

`rgbObjToHsl(rgbObj)`

Parameter	Type	Required	Description
`rgbObj`	RgbObj	true	An RGB object that represents a color in the RGBA (Red, Green, Blue, Alpha) color space.

Return Value

The `rgbObjToHsl` function returns an HSL object of the color value

Examples

A color with opacity

```
import { rgbObjToHsl } from '@snipshot/spectrum';

const hsl = rgbObjToHsl({ r: 138, g: 217, b: 16, a: 0.36 });

console.log(hsl); // { h: 84, s: 0.86, l: 0.46, a: 0.36 }
```

setHsl()

The `setHsl` function allows you to modify the HSL (Hue, Saturation, Lightness) values of a color object. This function returns a new `Spectrum` instance with the updated HSL values.

Usage

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

const color = new Spectrum('hsl', [180, 0.5, 0.32]);

const updatedColor = setHsl(color, {
  hue: 240, // Set hue equal to 240 degrees
  lightness: 0.7 // Set lightness equal to 70%
});

console.log(updatedColor.hsl); // { h: 240, s: 0.5, l: 0.7, a: 1 }
console.log(color.hex === updatedColor.hex); // false
```

Parameters

`setHsl(colorObj, options)`

Parameter	Type	Required	Valid range	Description
`colorObj`	Spectrum instance	true	-	The Spectrum instance representing the color you want to modify
`options.hue`	`number`	false	`[0; 360]`	The value that will be set as a hue value

Parameter	Type	Required	Valid range	Description
`options.saturation`	`number | string`	false	`[0; 1]` or `['0%'; '100%']`	The value that will be set as a saturation value
`options.lightness`	`number | string`	false	`[0; 1]` or `['0%'; '100%']`	The value that will be set as a lightness value
`options.alpha`	`number | string`	false	`[0; 1]` or `['0%'; '100%']`	The value that will be set as an alpha value

Return Value

The `setHsl` function returns a new `Spectrum` instance with the modified HSL values.

Examples

Modify all properties

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

const color = new Spectrum('hsl', [120, 0.7, 0.5, 1]);
const updatedColor = setHsl(color, {
  hue: 210,
  saturation: 0.35,
  lightness: 0.92,
  alpha: 0.9
});

console.log(updatedColor.hsl); // { h: 210, s: 0.35, l: 0.92, a: 0.9 }
```

Modify lightness

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

const color = new Spectrum('hsl', [60, 0.32, 0.48, 0.85]);
const updatedColor = setHsl(color, { lightness: 0.6 });

console.log(updatedColor.hsl); // { h: 60, s: 0.32, l: 0.6, a: 0.85 }
```

setRgb()

The `setRgb` function allows you to modify the RGB (Red, Green, Blue) values of a color object. This function returns a new `Spectrum` instance with the updated RGB values.

Usage

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

const color = new Spectrum('rgb', [255, 0, 0]);

const updatedColor = setRgb(color, {
  blue: 90, // Set blue channel equal to 90
  alpha: 0.7 // Set opacity equal to 70%
});

console.log(updatedColor.rgb); // { r: 255, g: 0, b: 90, a: 0.7 }
console.log(color.hex === updatedColor.hex); // false
```

Parameters

`setRgb(colorObj, options)`

Parameter	Type	Required	Valid range	Description
`colorObj`	Spectrum instance	true	-	The Spectrum instance representing the color you want to modify
`options.red`	`number`	false	`[0; 255]`	The value that will be set as a red channel value

Parameter	Type	Required	Valid range	Description
`options.green`	`number`	false	`[0; 255]`	The value that will be set as a green channel value
`options.blue`	`number`	false	`[0; 255]`	The value that will be set as a blue channel value
`options.alpha`	`number`	false	`[0; 1]` or `['0%'; '100%']`	The value that will be set as an alpha value

Return Value

The `setRgb` function returns a new `Spectrum` instance with the modified RGB values.

Examples

Modify all properties

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

const color = new Spectrum('rgb', [125, 240, 10, 0.35]);
const updatedColor = setRgb(color, {
  red: 210,
  green: 10,
  blue: 160,
  alpha: 1
});

console.log(updatedColor.rgb); // { r: 210, g: 10, b: 160, a: 1 }
```

Modify opacity

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

const color = new Spectrum('rgb', [240, 120, 128]);
const updatedColor = setRgb(color, { alpha: 0.65 });

console.log(updatedColor.rgb); // { r: 240, g: 120, b: 128, a: 0.65 }
```

Types

CssNamedColor

A `string` of a CSS named color, such as `'red'`, `'blue'`, `'black'` or `'lightseagreen'`. Can be used to initialize a `Spectrum` instance.

Type Definition

```
type CssNamedColor = "aliceblue" | "antiquewhite" | "aqua" | ... 145 more
... | "yellowgreen"
```

HslObj

An HSL object that represents a color in the HSLA (Hue, Saturation, Lightness, Alpha) color space.

Key	Description	Type	Valid range
`h`	Hue (color tone)	`number`	`[0; 360]`
`s`	Saturation (color intensity)	`number`	`[0; 1]`
`l`	Lightness (brightness)	`number`	`[0; 1]`
`a`	Alpha channel (opacity)	`number`	`[0; 1]`

Type Definition

```
type HslObj = {
  h: number;
  s: number;
  l: number;
  a: number;
};
```

RgbObj

An RGB object that represents a color in the RGBA (Red, Green, Blue, Alpha) color space.

Key	Description	Type	Valid range
r	Red	number	[0; 255]
g	Green	number	[0; 255]
b	Blue	number	[0; 255]
a	Alpha channel (opacity)	number	[0; 1]

Type Definition

```
type RgbObj = {
  r: number;
  g: number;
  b: number;
  a: number;
};
```

Contacts

If you have any questions, feedback, or need assistance with the Spectrum library, we're here to help. Feel free to reach out to us through any of the following channels:

GitHub Repository

You can find the official Spectrum library repository on GitHub. This is the place for bug reports, feature requests, and contributions to the library.

💂 Spectrum GitHub Repository

Email

Have a specific question or need direct assistance? Reach out to us via email, and we'll get back to you as soon as possible.

📧 Email: spectrum@snipshot.dev

Packages

Spectrum is available as both JSR and npm package, making it easy to include in your projects:

📦 Spectrum on **JSR**
📦 Spectrum on **npm**

We value your feedback and are committed to providing support to ensure you have a smooth experience with Spectrum. Don't hesitate to contact us: we're here to assist you.

Thank you for using Spectrum! 🎨