There's a newer version of Bootstrap!

Sass

Utilize our source Sass files to take advantage of variables, maps, mixins, and functions to help you build faster and customize your project.

Add passwordless login to your app in minutes. Passkeys, 2FA, Social Logins, & more. Start for free.

ads via Carbon

On this page

Utilize our source Sass files to take advantage of variables, maps, mixins, and more.

File structure

Whenever possible, avoid modifying Bootstrap's core files. For Sass, that means creating your own stylesheet that imports Bootstrap so you can modify and extend it. Assuming you're using a package manager like npm, you'll have a file structure that looks like this:

```
your-project/
├── scss
│   └── custom.scss
└── node_modules/
    └── bootstrap
        ├── js
        └── scss
```

If you've downloaded our source files and aren't using a package manager, you'll want to manually setup something similar to that structure, keeping Bootstrap's source files separate from your own.

```
your-project/
├── scss
│   └── custom.scss
└── bootstrap/
    ├── js
    └── scss
```

Importing

In your `custom.scss`, you'll import Bootstrap's source Sass files. You have two options: include all of Bootstrap, or pick the parts you need. We encourage the latter, though be aware there are some requirements and dependencies across our components. You also will need to include some JavaScript for our plugins.

```
// Custom.scss
// Option A: Include all of Bootstrap

// Include any default variable overrides here (though functions won't be available)

@import "../node_modules/bootstrap/scss/bootstrap";

// Then add additional custom code here
```

```scss
// Custom.scss
// Option B: Include parts of Bootstrap

// 1. Include functions first (so you can manipulate colors, SVGs, calc, etc)
@import "../node_modules/bootstrap/scss/functions";

// 2. Include any default variable overrides here

// 3. Include remainder of required Bootstrap stylesheets
@import "../node_modules/bootstrap/scss/variables";
@import "../node_modules/bootstrap/scss/mixins";

// 4. Include any optional Bootstrap components as you like
@import "../node_modules/bootstrap/scss/root";
@import "../node_modules/bootstrap/scss/reboot";
@import "../node_modules/bootstrap/scss/type";
@import "../node_modules/bootstrap/scss/images";
@import "../node_modules/bootstrap/scss/containers";
@import "../node_modules/bootstrap/scss/grid";

// 5. Add additional custom code here
```

With that setup in place, you can begin to modify any of the Sass variables and maps in your `custom.scss`. You can also start to add parts of Bootstrap under the `// Optional` section as needed. We suggest using the full import stack from our `bootstrap.scss` file as your starting point.

Variable defaults

Every Sass variable in Bootstrap includes the `!default` flag allowing you to override the variable's default value in your own Sass without modifying Bootstrap's source code. Copy and paste variables as needed, modify their values, and remove the `!default` flag. If a variable has already been assigned, then it won't be re-assigned by the default values in Bootstrap.

You will find the complete list of Bootstrap's variables in `scss/_variables.scss`. Some variables are set to `null`, these variables don't output the property unless they are overridden in your configuration.

Variable overrides must come after our functions are imported, but before the rest of the imports.

Here's an example that changes the `background-color` and `color` for the `<body>` when importing and compiling Bootstrap via npm:

```scss
// Required
@import "../node_modules/bootstrap/scss/functions";
```

```scss
// Default variable overrides
$body-bg: #000;
$body-color: #111;

// Required
@import "../node_modules/bootstrap/scss/variables";
@import "../node_modules/bootstrap/scss/mixins";

// Optional Bootstrap components here
@import "../node_modules/bootstrap/scss/root";
@import "../node_modules/bootstrap/scss/reboot";
@import "../node_modules/bootstrap/scss/type";
// etc
```

Repeat as necessary for any variable in Bootstrap, including the global options below.

> **Get started with Bootstrap via npm with our starter project!** Head to the twbs/bootstrap-npm-starter template repository to see how to build and customize Bootstrap in your own npm project. Includes Sass compiler, Autoprefixer, Stylelint, PurgeCSS, and Bootstrap Icons.

Maps and loops

Bootstrap includes a handful of Sass maps, key value pairs that make it easier to generate families of related CSS. We use Sass maps for our colors, grid breakpoints, and more. Just like Sass variables, all Sass maps include the `!default` flag and can be overridden and extended.

Some of our Sass maps are merged into empty ones by default. This is done to allow easy expansion of a given Sass map, but comes at the cost of making *removing* items from a map slightly more difficult.

Modify map

All variables in the `$theme-colors` map are defined as standalone variables. To modify an existing color in our `$theme-colors` map, add the following to your custom Sass file:

```scss
$primary: #0074d9;
$danger: #ff4136;
```

Later on, these variables are set in Bootstrap's `$theme-colors` map:

```
$theme-colors: (
  "primary": $primary,
  "danger": $danger
);
```

Add to map

Add new colors to `$theme-colors`, or any other map, by creating a new Sass map with your custom values and merging it with the original map. In this case, we'll create a new `$custom-colors` map and merge it with `$theme-colors`.

```
// Create your own map
$custom-colors: (
  "custom-color": #900
);

// Merge the maps
$theme-colors: map-merge($theme-colors, $custom-colors);
```

Remove from map

To remove colors from `$theme-colors`, or any other map, use `map-remove`. Be aware you must insert it between our requirements and options:

```
// Required
@import "../node_modules/bootstrap/scss/functions";
@import "../node_modules/bootstrap/scss/variables";
@import "../node_modules/bootstrap/scss/mixins";

$theme-colors: map-remove($theme-colors, "info", "light", "dark");

// Optional
@import "../node_modules/bootstrap/scss/root";
@import "../node_modules/bootstrap/scss/reboot";
@import "../node_modules/bootstrap/scss/type";
// etc
```

Required keys

Bootstrap assumes the presence of some specific keys within Sass maps as we used and extend these ourselves. As you customize the included maps, you may encounter errors where a specific Sass map's key is being used.

For example, we use the `primary`, `success`, and `danger` keys from `$theme-colors` for links, buttons, and form states. Replacing the values of these keys should present no issues, but removing them may cause Sass compilation issues. In these instances, you'll need to modify the Sass code that makes use of those values.

Functions

Colors

Next to the Sass maps we have, theme colors can also be used as standalone variables, like `$primary`.

```
.custom-element {
  color: $gray-100;
  background-color: $dark;
}
```

You can lighten or darken colors with Bootstrap's `tint-color()` and `shade-color()` functions. These functions will mix colors with black or white, unlike Sass' native `lighten()` and `darken()` functions which will change the lightness by a fixed amount, which often doesn't lead to the desired effect.

```
// Tint a color: mix a color with white
@function tint-color($color, $weight) {
  @return mix(white, $color, $weight);
}

// Shade a color: mix a color with black
@function shade-color($color, $weight) {
  @return mix(black, $color, $weight);
}

// Shade the color if the weight is positive, else tint it
@function shift-color($color, $weight) {
  @return if($weight > 0, shade-color($color, $weight), tint-color($color, -$weight)
}
```

In practice, you'd call the function and pass in the color and weight parameters.

```scss
.custom-element {
  color: tint-color($primary, 10%);
}

.custom-element-2 {
  color: shade-color($danger, 30%);
}
```

Color contrast

In order to meet WCAG 2.0 accessibility standards for color contrast, authors **must** provide a contrast ratio of at least 4.5:1, with very few exceptions.

An additional function we include in Bootstrap is the color contrast function, `color-contrast`. It utilizes the WCAG 2.0 algorithm for calculating contrast thresholds based on relative luminance in a `sRGB` colorspace to automatically return a light (`#fff`), dark (`#212529`) or black (`#000`) contrast color based on the specified base color. This function is especially useful for mixins or loops where you're generating multiple classes.

For example, to generate color swatches from our `$theme-colors` map:

```scss
@each $color, $value in $theme-colors {
  .swatch-#{$color} {
    color: color-contrast($value);
  }
}
```

It can also be used for one-off contrast needs:

```scss
.custom-element {
  color: color-contrast(#000); // returns `color: #fff`
}
```

You can also specify a base color with our color map functions:

```scss
.custom-element {
  color: color-contrast($dark); // returns `color: #fff`
```

```
}
```

Escape SVG

We use the `escape-svg` function to escape the `<`, `>` and `#` characters for SVG background images. When using the `escape-svg` function, data URIs must be quoted.

Add and Subtract functions

We use the `add` and `subtract` functions to wrap the CSS `calc` function. The primary purpose of these functions is to avoid errors when a "unitless" `0` value is passed into a `calc` expression. Expressions like `calc(10px - 0)` will return an error in all browsers, despite being mathematically correct.

Example where the calc is valid:

```
$border-radius: .25rem;
$border-width: 1px;

.element {
  // Output calc(.25rem - 1px) is valid
  border-radius: calc($border-radius - $border-width);
}

.element {
  // Output the same calc(.25rem - 1px) as above
  border-radius: subtract($border-radius, $border-width);
}
```

Example where the calc is invalid:

```
$border-radius: .25rem;
$border-width: 0;

.element {
  // Output calc(.25rem - 0) is invalid
  border-radius: calc($border-radius - $border-width);
}

.element {
  // Output .25rem
  border-radius: subtract($border-radius, $border-width);
```

```
}
```

Mixins

Our `scss/mixins/` directory has a ton of mixins that power parts of Bootstrap and can also be used across your own project.

Color schemes

A shorthand mixin for the `prefers-color-scheme` media query is available with support for `light`, `dark`, and custom color schemes.

```
@mixin color-scheme($name) {
  @media (prefers-color-scheme: #{$name}) {
    @content;
  }
}
```

```
.custom-element {
  @include color-scheme(dark) {
    // Insert dark mode styles here
  }

  @include color-scheme(custom-named-scheme) {
    // Insert custom color scheme styles here
  }
}
```

Bootstrap

Designed and built with all the love in the world by the Bootstrap team with the help of our contributors.

Code licensed MIT, docs CC BY 3.0.

Currently v5.0.2.

Analytics by Fathom.

Links

Home

Docs

Examples

Themes

Blog

Swag Store

Guides

Getting started

Starter template

Webpack

Parcel

Projects

Bootstrap 5

Bootstrap 4

Icons

RFS

npm starter

Community

Issues

Discussions

Corporate sponsors

Open Collective

Stack Overflow