Components

View on GitHub

Learn how and why we build nearly all our components responsively and with base and modifier classes.

Base classes

Bootstrap's components are largely built with a base-modifier nomenclature. We group as many shared properties as possible into a base class, like `.btn`, and then group individual styles for each variant into modifier classes, like `.btn-primary` or `.btn-success`.

To build our modifier classes, we use Sass's `@each` loops to iterate over a Sass map. This is especially helpful for generating variants of a component by our `$theme-colors` and creating responsive variants for each breakpoint. As you customize these Sass maps and recompile, you'll automatically see your changes reflected in these loops.

Check out our Sass maps and loops docs for how to customize these loops and extend Bootstrap's base-modifier approach to your own code.

Modifiers

Many of Bootstrap's components are built with a base-modifier class approach. This means the bulk of the styling is contained to a base class (e.g., `.btn`) while style variations are confined to modifier classes (e.g., `.btn-danger`). These modifier classes are built from the `$theme-colors` map to make customizing the number and name of our modifier classes.

Here are two examples of how we loop over the `$theme-colors` map to generate modifiers to the `.alert` and `.list-group` components.

Copy

```scss
// Generate contextual modifier classes for colorizing the alert.

@each $state, $value in $theme-colors {
  $alert-background: shift-color($value, $alert-bg-scale);
  $alert-border: shift-color($value, $alert-border-scale);
  $alert-color: shift-color($value, $alert-color-scale);
  @if (contrast-ratio($alert-background, $alert-color) < $min-contrast-ratio) {
    $alert-color: mix($value, color-contrast($alert-background), abs($alert-color-scale));
  }
  .alert-#{$state} {
    @include alert-variant($alert-background, $alert-border, $alert-color);
  }
}
```

Copy

```scss
// List group contextual variants
//
// Add modifier classes to change text and background color on individual items.
// Organizationally, this must come after the `:hover` states.

@each $state, $value in $theme-colors {
  $list-group-variant-bg: shift-color($value, $list-group-item-bg-scale);
  $list-group-variant-color: shift-color($value, $list-group-item-color-scale);
  @if (contrast-ratio($list-group-variant-bg, $list-group-variant-color) < $min-contrast-ratio) {
    $list-group-variant-color: mix($value, color-contrast($list-group-variant-bg), abs($list-group-ite
  }

  @include list-group-item-variant($state, $list-group-variant-bg, $list-group-variant-color);
}
```

Responsive

These Sass loops aren't limited to color maps, either. You can also generate responsive variations of your components. Take for example our responsive alignment of the dropdowns where we mix an `@each` loop for the `$grid-breakpoints` Sass map with a media query include.

```scss
// We deliberately hardcode the `bs-` prefix because we check
// this custom property in JS to determine Popper's positioning

@each $breakpoint in map-keys($grid-breakpoints) {
  @include media-breakpoint-up($breakpoint) {
    $infix: breakpoint-infix($breakpoint, $grid-breakpoints);

    .dropdown-menu#{$infix}-start {
      --bs-position: start;

      &[data-bs-popper] {
        right: auto;
        left: 0;
      }
    }

    .dropdown-menu#{$infix}-end {
      --bs-position: end;

      &[data-bs-popper] {
        right: 0;
        left: auto;
      }
    }
  }
}
```

Should you modify your `$grid-breakpoints`, your changes will apply to all the loops iterating over that map.

```scss
$grid-breakpoints: (
  xs: 0,
  sm: 576px,
  md: 768px,
  lg: 992px,
  xl: 1200px,
  xxl: 1400px
);
```

For more information and examples on how to modify our Sass maps and variables, please refer to the Sass section of the Grid documentation.

Creating your own

We encourage you to adopt these guidelines when building with Bootstrap to create your own components. We've extended this approach ourselves to the custom components in our documentation and examples. Components like our callouts are built just like our provided components with base and modifier classes.

> **This is a callout.** We built it custom for our docs so our messages to you stand out. It has three variants via modifier classes.

```html
<div class="callout">...</div>
```

In your CSS, you'd have something like the following where the bulk of the styling is done via `.callout`. Then, the unique styles between each variant is controlled via modifier class.

```scss
// Base class
.callout {}

// Modifier classes
.callout-info {}
.callout-warning {}
.callout-danger {}
```

For the callouts, that unique styling is just a `border-left-color`. When you combine that base class with one of those

modifier classes, you get your complete component family:

This is an info callout. Example text to show it in action.

This is a warning callout. Example text to show it in action.

This is a danger callout. Example text to show it in action.