Utility API

Contents

Bootstrap utilities are generated with our utility API and can be used to modify or extend our default set of utility classes via Sass. Our utility API is based on a series of Sass maps and functions for generating families of classes with various options. If you're unfamiliar with Sass maps, read up on the official Sass docs to get started.

The `$utilities` map contains all our utilities and is later merged with your custom `$utilities` map, if present. The utility map contains a keyed list of utility groups which accept the following options:

{{< bs-table "table text-start" >}} | Option | Type | Default value | Description | | — | — | — | — | | `property` | **Required** | – | Name of the property, this can be a string or an array of strings (e.g., horizontal paddings or margins). | | `values` | **Required** | – | List of values, or a map if you don't want the class name to be the same as the value. If `null` is used as map key, it isn't compiled. | | `class` | Optional | null | Name of the generated class. If not provided and `property` is an array of strings, `class` will default to the first element of the `property` array. | | `css-var` | Optional | `false` | Boolean to generate CSS variables instead of CSS rules. | | `local-vars` | Optional | null | Map of local

CSS variables to generate in addition to the CSS rules. | | `state` | Optional | null | List of pseudo-class variants (e.g., `:hover` or `:focus`) to generate. | | `responsive` | Optional | `false` | Boolean indicating if responsive classes should be generated. | | `rfs` | Optional | `false` | Boolean to enable [fluid rescaling with RFS]({{< docsref "/getting-started/rfs" >}}). | | `print` | Optional | `false` | Boolean indicating if print classes need to be generated. | | `rtl` | Optional | `true` | Boolean indicating if utility should be kept in RTL. | {{< /bs-table >}}

API explained

All utility variables are added to the `$utilities` variable within our `_utilities.scss` stylesheet. Each group of utilities looks something like this:

```
$utilities: (
  "opacity": (
    property: opacity,
    values: (
      0: 0,
      25: .25,
      50: .5,
      75: .75,
      100: 1,
    )
  )
);
```

Which outputs the following:

```
.opacity-0 { opacity: 0; }
.opacity-25 { opacity: .25; }
.opacity-50 { opacity: .5; }
.opacity-75 { opacity: .75; }
.opacity-100 { opacity: 1; }
```

Property

The required `property` key must be set for any utility, and it must contain a valid CSS property. This property is used in the generated utility's ruleset. When the `class` key is omitted, it also serves as the default class name. Consider the `text-decoration` utility:

```
$utilities: (
  "text-decoration": (
    property: text-decoration,
    values: none underline line-through
  )
);
```

Output:

```
.text-decoration-none { text-decoration: none !important; }
.text-decoration-underline { text-decoration: underline !important; }
.text-decoration-line-through { text-decoration: line-through !important; }
```

Values

Use the `values` key to specify which values for the specified `property` should be used in the generated class names and rules. Can be a list or map (set in the utilities or in a Sass variable).

As a list, like with [`text-decoration` utilities]({{< docsref "/utilities/text#text-decoration" >}}):

```
values: none underline line-through
```

As a map, like with [`opacity` utilities]({{< docsref "/utilities/opacity" >}}):

```
values: (
  0: 0,
  25: .25,
  50: .5,
  75: .75,
  100: 1,
)
```

As a Sass variable that sets the list or map, as in our [`position` utilities]({{< docsref "/utilities/position" >}}):

```
values: $position-values
```

Class

Use the `class` option to change the class prefix used in the compiled CSS. For example, to change from `.opacity-*` to `.o-*`:

```
$utilities: (
  "opacity": (
    property: opacity,
    class: o,
    values: (
      0: 0,
      25: .25,
      50: .5,
      75: .75,
      100: 1,
    )
  )
);
```

Output:

```
.o-0 { opacity: 0 !important; }
.o-25 { opacity: .25 !important; }
.o-50 { opacity: .5 !important; }
.o-75 { opacity: .75 !important; }
.o-100 { opacity: 1 !important; }
```

CSS variable utilities

Set the `css-var` boolean option to `true` and the API will generate local CSS variables for the given selector instead of the usual `property: value` rules. Consider our `.text-opacity-*` utilities:

```
$utilities: (
  "text-opacity": (
    css-var: true,
    class: text-opacity,
    values: (
      25: .25,
      50: .5,
      75: .75,
      100: 1
    )
  ),
);
```

Output:

```
.text-opacity-25 { --bs-text-opacity: .25; }
.text-opacity-50 { --bs-text-opacity: .5; }
.text-opacity-75 { --bs-text-opacity: .75; }
.text-opacity-100 { --bs-text-opacity: 1; }
```

Local CSS variables

Use the `local-vars` option to specify a Sass map that will generate local CSS variables within the utility class's ruleset. Please note that it may require additional work to consume those local CSS variables in the generated CSS rules. For example, consider our `.bg-*` utilities:

```
$utilities: (
  "background-color": (
    property: background-color,
    class: bg,
    local-vars: (
      "bg-opacity": 1
    ),
    values: map-merge(
      $utilities-bg-colors,
```

```
      (
        "transparent": transparent
      )
    )
  )
);
```

Output:

```
.bg-primary {
  --bs-bg-opacity: 1;
  background-color: rgba(var(--bs-primary-rgb), var(--bs-bg-opacity)) !important;
}
```

States

Use the `state` option to generate pseudo-class variations. Example pseudo-classes are `:hover` and `:focus`. When a list of states are provided, classnames are created for that pseudo-class. For example, to change opacity on hover, add `state: hover` and you'll get `.opacity-hover:hover` in your compiled CSS.

Need multiple pseudo-classes? Use a space-separated list of states: `state: hover focus`.

```
$utilities: (
  "opacity": (
    property: opacity,
    class: opacity,
    state: hover,
    values: (
      0: 0,
      25: .25,
      50: .5,
      75: .75,
      100: 1,
    )
  )
);
```

Output:

```
.opacity-0-hover:hover { opacity: 0 !important; }
.opacity-25-hover:hover { opacity: .25 !important; }
.opacity-50-hover:hover { opacity: .5 !important; }
.opacity-75-hover:hover { opacity: .75 !important; }
.opacity-100-hover:hover { opacity: 1 !important; }
```

Responsive

Add the `responsive` boolean to generate responsive utilities (e.g., `.opacity-md-25`) across [all breakpoints]({{< docsref "/layout/breakpoints" >}}).

```
$utilities: (
  "opacity": (
    property: opacity,
    responsive: true,
    values: (
      0: 0,
      25: .25,
      50: .5,
      75: .75,
      100: 1,
    )
  )
);
```

Output:

```
.opacity-0 { opacity: 0 !important; }
.opacity-25 { opacity: .25 !important; }
.opacity-50 { opacity: .5 !important; }
.opacity-75 { opacity: .75 !important; }
.opacity-100 { opacity: 1 !important; }

@media (min-width: 576px) {
  .opacity-sm-0 { opacity: 0 !important; }
  .opacity-sm-25 { opacity: .25 !important; }
  .opacity-sm-50 { opacity: .5 !important; }
  .opacity-sm-75 { opacity: .75 !important; }
  .opacity-sm-100 { opacity: 1 !important; }
}

@media (min-width: 768px) {
  .opacity-md-0 { opacity: 0 !important; }
  .opacity-md-25 { opacity: .25 !important; }
  .opacity-md-50 { opacity: .5 !important; }
  .opacity-md-75 { opacity: .75 !important; }
  .opacity-md-100 { opacity: 1 !important; }
}

@media (min-width: 992px) {
  .opacity-lg-0 { opacity: 0 !important; }
  .opacity-lg-25 { opacity: .25 !important; }
```

```
  .opacity-lg-50 { opacity: .5 !important; }
  .opacity-lg-75 { opacity: .75 !important; }
  .opacity-lg-100 { opacity: 1 !important; }
}

@media (min-width: 1200px) {
  .opacity-xl-0 { opacity: 0 !important; }
  .opacity-xl-25 { opacity: .25 !important; }
  .opacity-xl-50 { opacity: .5 !important; }
  .opacity-xl-75 { opacity: .75 !important; }
  .opacity-xl-100 { opacity: 1 !important; }
}

@media (min-width: 1400px) {
  .opacity-xxl-0 { opacity: 0 !important; }
  .opacity-xxl-25 { opacity: .25 !important; }
  .opacity-xxl-50 { opacity: .5 !important; }
  .opacity-xxl-75 { opacity: .75 !important; }
  .opacity-xxl-100 { opacity: 1 !important; }
}
```

Print

Enabling the `print` option will **also** generate utility classes for print, which are only applied within the `@media print { ... }` media query.

```
$utilities: (
  "opacity": (
    property: opacity,
    print: true,
    values: (
      0: 0,
      25: .25,
      50: .5,
      75: .75,
      100: 1,
    )
  )
);
```

Output:

```
.opacity-0 { opacity: 0 !important; }
.opacity-25 { opacity: .25 !important; }
.opacity-50 { opacity: .5 !important; }
.opacity-75 { opacity: .75 !important; }
.opacity-100 { opacity: 1 !important; }
```

```
@media print {
  .opacity-print-0 { opacity: 0 !important; }
  .opacity-print-25 { opacity: .25 !important; }
  .opacity-print-50 { opacity: .5 !important; }
  .opacity-print-75 { opacity: .75 !important; }
  .opacity-print-100 { opacity: 1 !important; }
}
```

Importance

All utilities generated by the API include `!important` to ensure they override components and modifier classes as intended. You can toggle this setting globally with the `$enable-important-utilities` variable (defaults to `true`).

Using the API

Now that you're familiar with how the utilities API works, learn how to add your own custom classes and modify our default utilities.

Override utilities

Override existing utilities by using the same key. For example, if you want additional responsive overflow utility classes, you can do this:

```
$utilities: (
  "overflow": (
    responsive: true,
    property: overflow,
    values: visible hidden scroll auto,
  ),
);
```

Add utilities

New utilities can be added to the default `$utilities` map with a `map-merge`. Make sure our required Sass files and `_utilities.scss` are imported first, then use the `map-merge` to add your additional utilities. For example, here's how to add a responsive `cursor` utility with three values.

```
@import "bootstrap/scss/functions";
@import "bootstrap/scss/variables";
@import "bootstrap/scss/utilities";

$utilities: map-merge(
  $utilities,
  (
```

```
    "cursor": (
      property: cursor,
      class: cursor,
      responsive: true,
      values: auto pointer grab,
    )
  )
);
```

Modify utilities

Modify existing utilities in the default `$utilities` map with `map-get` and `map-merge` functions. In the example below, we're adding an additional value to the `width` utilities. Start with an initial `map-merge` and then specify which utility you want to modify. From there, fetch the nested `"width"` map with `map-get` to access and modify the utility's options and values.

```
@import "bootstrap/scss/functions";
@import "bootstrap/scss/variables";
@import "bootstrap/scss/utilities";

$utilities: map-merge(
  $utilities,
  (
    "width": map-merge(
      map-get($utilities, "width"),
      (
        values: map-merge(
          map-get(map-get($utilities, "width"), "values"),
          (10: 10%),
        ),
      ),
    ),
  )
);
```

Enable responsive You can enable responsive classes for an existing set of utilities that are not currently responsive by default. For example, to make the `border` classes responsive:

```
@import "bootstrap/scss/functions";
@import "bootstrap/scss/variables";
@import "bootstrap/scss/utilities";

$utilities: map-merge(
  $utilities, (
    "border": map-merge(
```

```
      map-get($utilities, "border"),
      ( responsive: true ),
    ),
  )
);
```

This will now generate responsive variations of `.border` and `.border-0` for each breakpoint. Your generated CSS will look like this:

```
.border { ... }
.border-0 { ... }

@media (min-width: 576px) {
  .border-sm { ... }
  .border-sm-0 { ... }
}

@media (min-width: 768px) {
  .border-md { ... }
  .border-md-0 { ... }
}

@media (min-width: 992px) {
  .border-lg { ... }
  .border-lg-0 { ... }
}

@media (min-width: 1200px) {
  .border-xl { ... }
  .border-xl-0 { ... }
}

@media (min-width: 1400px) {
  .border-xxl { ... }
  .border-xxl-0 { ... }
}
```

Rename utilities Missing v4 utilities, or used to another naming convention? The utilities API can be used to override the resulting `class` of a given utility—for example, to rename `.ms-*` utilities to oldish `.ml-*`:

```
@import "bootstrap/scss/functions";
@import "bootstrap/scss/variables";
@import "bootstrap/scss/utilities";

$utilities: map-merge(
  $utilities, (
```

```
    "margin-start": map-merge(
      map-get($utilities, "margin-start"),
      ( class: ml ),
    ),
  )
);
```

Remove utilities

Remove any of the default utilities by setting the group key to `null`. For example, to remove all our `width` utilities, create a `$utilities` `map-merge` and add `"width": null` within.

```
@import "bootstrap/scss/functions";
@import "bootstrap/scss/variables";
@import "bootstrap/scss/utilities";

$utilities: map-merge(
  $utilities,
  (
    "width": null
  )
);
```

Remove utility in RTL Some edge cases make RTL styling difficult, such as line breaks in Arabic. Thus utilities can be dropped from RTL output by setting the `rtl` option to `false`:

```
$utilities: (
  "word-wrap": (
    property: word-wrap word-break,
    class: text,
    values: (break: break-word),
    rtl: false
  ),
);
```

Output:

```
/* rtl:begin:remove */
.text-break {
  word-wrap: break-word !important;
  word-break: break-word !important;
}
/* rtl:end:remove */
```

This doesn't output anything in RTL, thanks to the RTLCSS `remove` control directive.