Search 6.4.0 docs

Docs Menu 6.4.0

Customizing Icons

Font Awesome icons can be customized even further using your own CSS. We've even added CSS Custom Properties to our style toolkit options.

Before You Get Started

Make sure you're:

- **Set up with Font Awesome** in your project.
- Familiar with the basics of **adding Font Awesome icons**.

Customize with CSS Custom Properties

Our styling toolkit provides a lot of utility, including size, rotate, stack, and animate icons. Using the **CSS custom properties** below, we've added easy ways to customize aspects of our styling toolkit's features.

CSS Custom Property	Details	Accepted Values
`--fa-style-family`	Set Font Awesome icon family	`"Font Awesome 6 Free"` `"Font Awesome 6 Pro"` `"Font Awesome 6 Duotone"` `"Font Awesome 6 Brands"` `"Font Awesome 6 Sharp"`
`--fa-style`	Set Font Awesome icon style	Any valid Font Awesome **style weight**
`--fa-display`	Set display of an icon	Any valid CSS `display` **value**
`--fa-inverse`	Set color of an inverted icon	Any valid CSS `color` **value**
Icons in a List		
`--fa-li-margin`	Set margin around icon	Any valid CSS `margin` **value**
`--fa-li-width`	Set width of icon	Any valid CSS `width` **value**
Rotating Icons		
`--fa-rotate-angle`	Set rotation angle of `.fa-rotate-by`	Any valid CSS `transform` rotate **value**
Animating Icons		

CSS Custom Property	Details	Accepted Values
`--fa-animation-delay`	Set an initial delay for animation	Any valid CSS `animation-direction` **value**
`--fa-animation-direction`	Set direction for animation	Any valid CSS `animation-direction` **value**
`--fa-animation-duration`	Set duration for animation	Any valid CSS `animation-duration` **value**
`--fa-animation-iteration-count`	Set number of iterations for animation	Any valid CSS `animation-iteration-count` **value**
`--fa-animation-timing`	Set how the animation progresses through frames	Any valid CSS `animation-timing-function` **value**
`--fa-beat-scale`	Set the max value an `fa-beat` icon will scale	Any valid CSS number **value**
`--fa-fade-opacity`	Set lowest opacity value an `fa-fade` icon will fade to	`0` ↔ `1.0`
`--fa-beat-fade-opacity`	Set lowest opacity value an `fa-beat-fade` icon will fade to and from	`0` ↔ `1.0`
`--fa-beat-fade-scale`	Set max value that an icon will scale	Set the max value an `fa-beat-fade` icon will scale
`--fa-flip-x`	Set an `fa-flip` icon's x-coordinate of the vector denoting the axis of rotation	Any valid CSS `number` **value** between `0` and `1`
`--fa-flip-y`	Set an `fa-flip` icon'sy-coordinate of the vector denoting the axis of rotation	Any valid CSS `number` **value** between `0` and `1`
`--fa-flip-z`	Set an `fa-flip` icon's z-coordinate of the vector denoting the axis of rotation	Any valid CSS `number` **value** between `0`

CSS Custom Property	Details	Accepted Values
		and `1`
`--fa-flip-angle`	Set an `fa-flip` icon's rotation angle. A positive angle denotes a clockwise rotation, a negative angle a counter-clockwise one.	Any valid CSS angle **value**
Bordered Icons		
`--fa-border-color`	Set border color	Any valid CSS `border-color` **value**
`--fa-border-padding`	Set padding around icon	Any valid CSS `padding` **value**
`--fa-border-radius`	Set border radius	Any valid CSS `border-radius` **value**
`--fa-border-style`	Set border style	Any valid CSS `border-style` **value**
`--fa-border-width`	Set border width	Any valid CSS `border-width` **value**
Pulled Icons		
`--fa-pull-margin`	Set margin around icon	Any valid CSS `margin` **value**
Stacking Icons		
`--fa-stack-z-index`	Set z-index of a stacked icon	Any valid CSS `z-index` **value**
`--fa-inverse`	Set color of an inverted stacked icon	Any valid CSS `color` **value**

CSS Custom Property	Details	Accepted Values
Duotone Icons		
`--fa-primary-color`	Set primary layer color	Any valid CSS `color` **value**
`--fa-primary-opacity`	Set primary layer opacity	`0` ↔ `1.0`
`--fa-secondary-color`	Set secondary layer color	Any valid CSS `color` **value**
`--fa-secondary-opacity`	Set secondary layer opacity	`0` ↔ `1.0`
Layering Text and Counters		
`--fa-counter-background-color`	Set `fa-layers-counter`'s background color	Any valid CSS `color` **value**
`--fa-counter-border-radius`	Set `fa-layers-counter`'s border radius	Any valid CSS `border-radius` **value**
`--fa-counter-line-height`	Set `fa-layers-counter`'s line-height	Any valid CSS `line-height` **value**
`--fa-counter-max-width`	Set `fa-layers-counter`'s max-width	Any valid CSS `width` **value**
`--fa-counter-min-width`	Set `fa-layers-counter`'s min-width	Any valid CSS `width` **value**
`--fa-counter-padding`	Set `fa-layers-counter`'s padding	Any valid CSS `padding` **value**
`--fa-counter-scale`	Set how much `fa-layers-counter` is scaled up/down	Any valid CSS `transform` scale **value**

CSS Custom Property	Details	Accepted Values
`--fa-top`	Set top offset of `.fa-layers-top-left` or `layers-top-right`	Any valid CSS `top` **value**
`--fa-right`	Set right offset of `.fa-layers-top-right` or `layers-bottom-right`	Any valid CSS `right` **value**
`--fa-bottom`	Set bottom offset of `.fa-layers-bottom-left` or `layers-bottom-right`	Any valid CSS `bottom` **value**
`--fa-left`	Set left offset of `.fa-layers-top-left` or `layers-bottom-left`	Any valid CSS `left` **value**
`--fa-inverse`	Set color of an inverted stacked icon	Any valid CSS `color` **value**
Pseudo-elements		
`--fa-font-solid`	Sets the `font` property to use the solid style (`font-family` and `font-weight`) in pseudo-element custom CSS rules	
`--fa-font-regular`	Sets the `font` property to use the regular style (`font-family` and `font-weight`) in pseudo-element custom CSS rules	
`--fa-font-light`	Sets the `font` property to use the solid style (`font-family` and `font-weight`) in pseudo-element custom CSS rules	
`--fa-font-thin`	Sets the `font` property to use the thin style (`font-family` and `font-weight`) in pseudo-element custom CSS rules	
`--fa-font-duotone`	Sets the `font` property to use the duotone style (`font-family` and `font-weight`) in pseudo-element custom CSS rules	
`--fa-font-brands`	Sets the `font` property to use the brands style (`font-family` and `font-weight`) in pseudo-	

CSS Custom Property	Details	Accepted Values
	element custom CSS rules	
`--fa-font-sharp-solid`	Sets the `font` property to use the sharp family and solid style (`font-family` and `font-weight`) in pseudo-element custom CSS rules	
`--fa-font-sharp-regular`	Sets the `font` property to use the sharp family and regular style (`font-family` and `font-weight`) in pseudo-element custom CSS rules	
`--fa-font-sharp-light`	Sets the `font` property to use the sharp family and light style (`font-family` and `font-weight`) in pseudo-element custom CSS rules	
Fast Style Switching		
`--fa-style-family-classic`	Sets the `font-family` to `"Font Awesome 6 Pro"` in **Pro** projects) Sets the `font-family` `"Font Awesome 6 Free"` in **Free** projects.	
`--fa-style-family-sharp`	Sets the `font-family` to `"Font Awesome 6 Sharp"` in **Pro** projects)	

Using Custom Properties in a Project

CSS custom properties are still a pretty new thing for most folks. Here are some ways you can define them within your project...

Setting Properties with CSS :root

You can define custom properties at **CSS :root pseudo-class** level. This will make any icons that use a styling toolkit's feature inherit the properties by default.

```
<!-- by default, everything will be in v6's Classic regular style and in the
<style>
```

```
  :root {
    --fa-style-family: "Font Awesome 6 Pro";
    --fa-style: 400;
    --fa-border-color: red;
    --fa-primary-color: green;
    --fa-secondary-color: red;
  }
</style>
```

Setting Properties with Project-Based CSS Rules

You can also set custom properties inside of your project's CSS, in the `<head>` of a page or in a separate stylesheet. These properties will be scoped to just elements that match the selector of the rule you've included them in.

```
<style>
  /* setting a decorative icon dropcap before a block of text */
  .ye-olde-icon-dropcap {
    --fa-border-color: WhiteSmoke;
    --fa-border-padding: 2em;
    --fa-border-radius: 0.25em;
    --fa-pull-margin: 2em;

    font-size: 8em;
  }

  /* playing DJ with fa-record-vinyl */
  .track-quick-spin {
    --fa-spin-duration: 0.25s;
    --fa-spin-iteration-count: 1;
    --fa-spin-timing: ease-out;
  }

  .track-vocals {
    --fa-spin-duration: 10s;
    --fa-spin-timing: ease-in-out;
    --fa-spin-iteration-count: 2;
  }

  .track-dope-hook {
```

```
    --fa-spin-duration: 10s;
    --fa-spin-iteration-count: 10;
  }

  /* Mo' Malfoy! Mo' Malfoy! Mo' Malfoy! */
  .theme-slytherin {
    --fa-primary-color: darkgreen;
    --fa-secondary-color: silver;
  }

  /* extra sharp flavor */
  .theme-sharp {
    font: var(--fa-font-sharp-solid);
  }
</style>
```

Setting Properties with Inline Styles

Many of the examples in these docs define CSS custom properties using inline styling by adding a `style` attribute to an element. This is best for one-offs or very custom colored/styled duotone icons that you won't need to change at a system level.

```
<!-- rotating an icon -->
<i class="fa-solid fa-bomb fa-rotate-by" style="--fa-rotate-angle: 45deg;"><

<!-- using inline styles to define duotone icon custom properties -->
<i class="fa-duotone fa-crow" style="--fa-primary-color: dodgerblue; --fa-s
```

Be careful of specificity and cascade!

CSS Custom properties that are redefined will trump each other. Defining a property in `:root {}` and then defining the same property in a rule or inline will result in the `:root` style being overridden.

Add Your Own Custom Styling with CSS

Everything you can typically control with CSS is up for grabs — from color to display to alignment. We recommend targeting icons in your CSS in a couple of different ways. You can also add your own custom styling to Font Awesome icons by adding your own CSS rules in your project's code.

Styling Case	Recommended Selector

Custom styling all icons	Add a consistent custom class to all icons (e.g. `.icon`, `.[projectprefix]-icon`, or `.fa-icon`)
	
	You can also use **style-classes** for the in-use icon styles
	`.fa-solid { ... }`
	`.fa-regular { ... }`
	`.fa-light { ... }`
	`.fa-thin { ... }`
	`.fa-duotone { ... }`
	`.fa-brands { ... }`
	`.fa-sharp-solid { ... }`

Custom styling a specific icon	Use the individual icon name, prefixed with `.fa-`
	`.fa-user { ... }`
	`.fa-font-awesome { ... }`

Here's a quick example of using those selectors to add custom color to Font Awesome icons:

```
<!-- reference your copy of Font Awesome in the head -->
<head>
  <link href="/your-path-to-fontawesome/css/fontawesome.css" rel="stylesheet
  <link href="/your-path-to-fontawesome/css/brands.css" rel="stylesheet">
  <link href="/your-path-to-fontawesome/css/solid.css" rel="stylesheet">
  <link href="/your-path-to-fontawesome/css/sharp-solid.css" rel="stylesheet
</head>
```

```
<!-- Add some style -->
<style>
  /* custom styling for all icons */
  .fa-solid,
  .fa-brands,
  .fa-sharp-solid {
    background-color: papayawhip;
    border-radius: 0.2em;
    padding: 0.3em;
  }

  /* custom styling for specific icons */
  .fa-fish {
    color: salmon;
  }

  .fa-frog {
    color: green;
  }

  .fa-user-ninja.vanished {
    opacity: 0.2;
  }

  .fa-facebook {
    color: rgb(59, 91, 152);
  }
</style>

<!-- Add some Icons -->
<body>
  <i class="fa-solid fa-fish"></i>
  <i class="fa-solid fa-frog"></i>
  <i class="fa-solid fa-user-ninja vanished"></i>
  <i class="fa-brands fa-facebook"></i>
  <i class="fa-sharp fa-solid fa-fish"></i>
</body>
```

Writing Custom CSS with our SVG + JS framework

When using our SVG framework, remember that Font Awesome-based `<i>` DOM elements are replaced with new `<svg>` elements by default. Be sure that your CSS rules target the right elements.