Building a button component

- Overview
- Markup
- Accessibility
  - Hover and focus together
  - Interactive focus ring
  - Ensuring passing color contrast
  - Hiding icons from folks who can't see
- Styles
  - An adaptive custom property strategy
  - Preparing for design consistency
    - The shared selector
    - Button accent color
    - Button text color
    - Button background color
    - Button background well
    - Button padding
    - Button border
    - Button hover highlight effect
    - Button text shadow
    - Button icon
    - Button shadow
    - Button transitions
    - All properties together in the selector
    - Dark theme adaptations
    - Reduced motion adaptations
    - A few shared styles
  - Styling buttons
    - Selector adjustment

- Cursor and touch adjustments
- Colors and borders
- Shadows
- Layout
- Spacing
- Touch and mouse UX
- Transitions
- Icons
- Customizing submit buttons
- Customize reset buttons
- Customize disabled buttons
- Customizing file input buttons
- Special dark theme exceptions
- Creating variants
  - Vibrant button
  - Large button
  - Icon button
- Conclusion
- Community remixes
  - Resources

In this post I want to share my thoughts on how to build a color-adaptive, responsive, and accessible `<button>` element. Try the demo and view the source!

> {% Video
> src="video/vS06HQ1YTsbMKSFTIPl2iogUQP73/EU3ZVqNDtCZEmaLCmZFn.mp4",
> autoplay="true", loop="true", muted="true" %}
> Buttons are interacted with via keyboard and mouse in the light and dark themes.

If you prefer video, here's a YouTube version of this post:

{% YouTube 'aJNq-b1zlog' %}

Overview

{% BrowserCompat 'html.elements.button' %}

The `<button>` element is built for user interaction. Its `click` event triggers from keyboard, mouse, touch, voice, and more, with smart rules about its timing. It also comes with some default styles in each browser, so you can use them directly without any customization. Use `color-scheme` to opt into browser-provided light and dark buttons too.

{% Codepen { user: 'web-dot-dev', id: 'PoQqQaW', height: 500, tab: 'result' } %}

There are also different types of buttons, each shown in the preceding Codepen embed. A `<button>` without a type will adapt to being within a `<form>`, changing to the submit type.

```
<!-- buttons -->
<button></button>
<button type="submit"></button>
<button type="button"></button>
<button type="reset"></button>

<!-- button state -->
<button disabled></button>

<!-- input buttons -->
<input type="button" />
<input type="file">
```

In this month's GUI Challenge, each button will get styles to help visually differentiate their intent. Reset buttons will have warning colors since they're destructive, and submit buttons will get blue accent text so they appear slightly more promoted than regular buttons.

> {% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/RHjiEG4X1lAQiLtCA105.png", alt="Preview of the final set of all button types, shown in a form and not in a form, with nice additions for icon buttons and customized buttons.", width="800", height="539" %}
> Preview of the final set of all button types, shown in a form and not in a form, with nice additions for icon buttons and customized buttons

{% Aside %} The buttons have always acted in their own distinct ways, so why not style them to support their behavior? {% endAside %}

Buttons also have pseudo classes for CSS to use for styling. These classes provide CSS hooks into customizing the feel of the button: `:hover` for when a mouse is over the button, `:active` for when a mouse or keyboard is pressing, and `:focus` or `:focus-visible` for assisting in assistive technology styling.

```
button:hover {}
button:active {}
button:focus {}
button:focus-visible {}
```

{% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/ip7GrYSsMo7dgEGpxUCg.png", alt="Preview of the final set of all button types in the dark theme.", width="800", height="539" %}
Preview of the final set of all button types in the dark theme

Markup

In addition to the button types provided by the HTML specification, I've added a button with an icon and a button with a custom class `btn-custom`.

```
<button>Default</button>
<input type="button" value="<input>"/>
<button>
  <svg viewBox="0 0 24 24" width="24" height="24" aria-hidden="true">
    <path d="..." />
  </svg>
  Icon
</button>
<button type="submit">Submit</button>
<button type="button">Type Button</button>
<button type="reset">Reset</button>
<button disabled>Disabled</button>
<button class="btn-custom">Custom</button>
<input type="file">
```

Then, for testing, each button is placed inside of a form. This way I can ensure styles are updated appropriately for the default button, which behaves as a submit button. I also switch the icon strategy, from inline SVG to a masked SVG, to ensure both work equally well.

```
<form>
  <button>Default</button>
  <input type="button" value="<input>"/>
  <button>Icon <span data-icon="cloud"></span></button>
  <button type="submit">Submit</button>
```

```
  <button type="button">Type Button</button>
  <button type="reset">Reset</button>
  <button disabled>Disabled</button>
  <button class="btn-custom btn-large" type="button">Large Custom</button>
  <input type="file">
</form>
```

The matrix of combinations is pretty overwhelming at this point. Between button types, pseudo-classes, and being in or out of a form, there are over 20 combinations of buttons. It's a good thing CSS can help us articulate each of them clearly!

Accessibility

Button elements are naturally accessible but there are a few common enhancements.

Hover and focus together

I like to group `:hover` and `:focus` together with the `:is()` functional pseudo selector. This helps ensure my interfaces always consider keyboard and assistive technology styles.

```
button:is(:hover, :focus) {
  …
}
```

> {% Video src="video/vS06HQ1YTsbMKSFTIPl2iogUQP73/FwNnm4ExQWz2FO1GVa7j.mp4", autoplay="true", loop="true", muted="true" %}
>
> Try a demo!

Interactive focus ring

I like to animate the focus ring for keyboard and assistive technology users. I accomplish this by animating the outline away from the button by 5px, but only when the button is not active. This creates an effect that makes the focus ring shrink back to the button size when pressed.

```
:where(button, input):where(:not(:active)):focus-visible {
  outline-offset: 5px;
}
```

{% Video src="video/vS06HQ1YTsbMKSFTIPl2iogUQP73/iMI3C1a3ktt1f3vZGscB.mp4", autoplay="true", loop="true", muted="true" %}

Ensuring passing color contrast

There are at least four different color combinations across light and dark that need consideration of color contrast: button, submit button, reset button, and disabled button. VisBug is used here to inspect and show all their scores at once:

{% Video src="video/vS06HQ1YTsbMKSFTIPl2iogUQP73/Wg0EzKQgCHMJin5ujYus.mp4", autoplay="true", loop="true", muted="true" %}

Hiding icons from folks who can't see

When creating an icon button, the icon should provide visual support to the button text. This also means the icon is not valuable to someone with sight loss. Fortunately the browser provides a way to hide items from screen-reader technology so people with sight loss aren't bothered with decorative button images:

```
<button>
  <svg … aria-hidden="true">...</svg>
  Icon Button
</button>
```

{% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/WCww9HPVdcaoJNLQB8Sq.png", alt="Chrome DevTools showing the accessibility tree for the button. The tree ignores the button image because it has aria-hidden set to true.", width="800", height="515" %}
Chrome DevTools showing the accessibility tree for the button. The tree ignores the button image because it has aria-hidden set to true

Styles

In this next section, I first establish a custom property system for managing the adaptive styles of the button. With those custom properties I can begin to select elements and customize their appearance.

An adaptive custom property strategy

The custom property strategy used in this GUI Challenge is very similar to that used in building a color scheme. For an adaptive light and dark color system, a custom property for each theme is defined and named accordingly. Then a single custom property is used to hold the current value of the theme and is assigned to a CSS property. Later, the single custom property can be updated to a different value, then updating the button style.

```
button {
  --_bg-light: white;
  --_bg-dark: black;
  --_bg: var(--_bg-light);

  background-color: var(--_bg);
}

@media (prefers-color-scheme: dark) {
  button {
    --_bg: var(--_bg-dark);
  }
}
```

{% Aside %} The underscore convention is from Lea Verou in this post Custom properties with defaults. {% endAside %}

What I like is that the light and dark themes are declarative and clear. The indirection and abstraction are offloaded into the `--_bg` custom property, which is now the only "reactive" property; `--_bg-light` and `--_bg-dark` are static. It's also clear to read that the light theme is the default theme and dark is only applied conditionally.

Preparing for design consistency

The shared selector

The following selector is used to target all the various types of buttons and is a bit overwhelming at first. `:where()` is used so customizing the button requires no specificity. Buttons are often adapted for alternative scenarios and the `:where()` selector ensures that task is easy. Inside `:where()`, each button type is selected, including the `::file-selector-button`, which can't be used inside of `:is()` or `:where()`.

```
:where(
  button,
```

```
  input[type="button"],
  input[type="submit"],
  input[type="reset"],
  input[type="file"]
),
:where(input[type="file"])::file-selector-button {
  …
}
```

All the custom properties will be scoped inside this selector. Time to review all the custom properties! There are quite a few custom properties used in this button. I'll describe each group as we go, then share the dark and reduced motion contexts at the end of the section.

Button accent color

Submit buttons and icons are a great place for a pop of color:

```
--_accent-light: hsl(210 100% 40%);
--_accent-dark: hsl(210 50% 70%);
--_accent: var(--_accent-light);
```

Button text color

Button text colors aren't white or black, they're darkened or lightened versions of `--_accent` using `hsl()` and sticking to the hue `210`:

```
--_text-light: hsl(210 10% 30%);
--_text-dark: hsl(210 5% 95%);
--_text: var(--_text-light);
```

Button background color

Button backgrounds follow the same `hsl()` pattern except for the light theme buttons—those are set to white so their surface makes them appear close to the user, or in front of other surfaces:

```
--_bg-light: hsl(0 0% 100%);
--_bg-dark: hsl(210 9% 31%);
--_bg: var(--_bg-light);
```

Button background well

This background color is for making a surface appear behind other surfaces, useful for the background of the file input:

```
--_input-well-light: hsl(210 16% 87%);
--_input-well-dark: hsl(204 10% 10%);
--_input-well: var(--_input-well-light);
```

Button padding

The spacing around the text in the button is done using the `ch` unit, a relative length to the font size. This becomes critical when large buttons are able to simply bump up the `font-size` and button scales proportionally:

```
--_padding-inline: 1.75ch;
--_padding-block: .75ch;
```

Button border

The button border radius is stashed into a custom property so the file input can match the other buttons. The border colors follow the established adaptive color system:

```
--_border-radius: .5ch;


--_border-light: hsl(210 14% 89%);
--_border-dark: var(--_bg-dark);
--_border: var(--_border-light);
```

Button hover highlight effect

These properties establish a size property for transitioning on interaction, and the highlight color follows the adaptive color system. We'll cover how these interact later in this post, but ultimately these are used for a `box-shadow` effect:

```
--_highlight-size: 0;


--_highlight-light: hsl(210 10% 71% / 25%);
--_highlight-dark: hsl(210 10% 5% / 25%);
--_highlight: var(--_highlight-light);
```

Button text shadow

Each button has a subtle text shadow style. This helps the text sit on top of the button, improving legibility and adding a nice layer of presentation polish.

```
--_ink-shadow-light: 0 1px 0 var(--_border-light);
--_ink-shadow-dark: 0 1px 0 hsl(210 11% 15%);
--_ink-shadow: var(--_ink-shadow-light);
```

Button icon

Icons are the size of two characters thanks to the relative length `ch` unit again, which will help the icon scale proportionally to the button text. The icon color leans on the `--_accent-color` for an adaptive and within-theme color.

```
--_icon-size: 2ch;
--_icon-color: var(--_accent);
```

{% Aside %} By having its own property, customizing the icon color can be done without changing the accent color. {% endAside %}

Button shadow

For shadows to properly adapt to light and dark, they need to both shift their color and opacity. Light theme shadows are best when they are subtle and tinted towards the surface color they overlay. Dark theme shadows need to be darker and more saturated so they can overlay darker surface colors.

```
--_shadow-color-light: 220 3% 15%;
--_shadow-color-dark: 220 40% 2%;
--_shadow-color: var(--_shadow-color-light);

--_shadow-strength-light: 1%;
--_shadow-strength-dark: 25%;
--_shadow-strength: var(--_shadow-strength-light);
```

With adaptive colors and strengths I can assemble two depths of shadows:

```
--_shadow-1: 0 1px 2px -1px hsl(var(--_shadow-color)/calc(var(--_shadow-strength) + 9%));

--_shadow-2:
  0 3px 5px -2px hsl(var(--_shadow-color)/calc(var(--_shadow-strength) + 3%)),
  0 7px 14px -5px hsl(var(--_shadow-color)/calc(var(--_shadow-strength) + 5%));
```

Furthermore, to give the buttons a slightly 3D appearance, a `1px` box-shadow creates the illusion:

```
--_shadow-depth-light: 0 1px var(--_border-light);
--_shadow-depth-dark: 0 1px var(--_bg-dark);
--_shadow-depth: var(--_shadow-depth-light);
```

Button transitions

Following the pattern for adaptive colors, I create two static properties to hold the design system options:

```
--_transition-motion-reduce: ;
--_transition-motion-ok:
  box-shadow 145ms ease,
  outline-offset 145ms ease
;
--_transition: var(--_transition-motion-reduce);
```

{% Aside %} In early designs of the buttons, I animated `border-color`, which was in the `--_transition-motion-reduce` property. I later removed it but left the pattern in case I changed my mind again. {% endAside %}

All properties together in the selector

{% Details %}

{% DetailsSummary %} All custom properties in a selector {% endDetailsSummary %}

```
:where(
  button,
  input[type="button"],
  input[type="submit"],
  input[type="reset"],
  input[type="file"]
),
:where(input[type="file"])::file-selector-button {
  --_accent-light: hsl(210 100% 40%);
  --_accent-dark: hsl(210 50% 70%);
  --_accent: var(--_accent-light);

  --_text-light: hsl(210 10% 30%);
  --_text-dark: hsl(210 5% 95%);
```

```
--_text: var(--_text-light);

--_bg-light: hsl(0 0% 100%);
--_bg-dark: hsl(210 9% 31%);
--_bg: var(--_bg-light);

--_input-well-light: hsl(210 16% 87%);
--_input-well-dark: hsl(204 10% 10%);
--_input-well: var(--_input-well-light);

--_padding-inline: 1.75ch;
--_padding-block: .75ch;

--_border-radius: .5ch;
--_border-light: hsl(210 14% 89%);
--_border-dark: var(--_bg-dark);
--_border: var(--_border-light);

--_highlight-size: 0;
--_highlight-light: hsl(210 10% 71% / 25%);
--_highlight-dark: hsl(210 10% 5% / 25%);
--_highlight: var(--_highlight-light);

--_ink-shadow-light: 0 1px 0 hsl(210 14% 89%);
--_ink-shadow-dark: 0 1px 0 hsl(210 11% 15%);
--_ink-shadow: var(--_ink-shadow-light);

--_icon-size: 2ch;
--_icon-color-light: var(--_accent-light);
--_icon-color-dark: var(--_accent-dark);
--_icon-color: var(--accent, var(--_icon-color-light));

--_shadow-color-light: 220 3% 15%;
--_shadow-color-dark: 220 40% 2%;
--_shadow-color: var(--_shadow-color-light);
--_shadow-strength-light: 1%;
--_shadow-strength-dark: 25%;
--_shadow-strength: var(--_shadow-strength-light);
--_shadow-1: 0 1px 2px -1px hsl(var(--_shadow-color)/calc(var(--_shadow-strength) + 9%));
--_shadow-2:
```

```
    0 3px 5px -2px hsl(var(--_shadow-color)/calc(var(--_shadow-strength) + 3%)),
    0 7px 14px -5px hsl(var(--_shadow-color)/calc(var(--_shadow-strength) + 5%))
  ;

  --_shadow-depth-light: hsl(210 14% 89%);
  --_shadow-depth-dark: var(--_bg-dark);
  --_shadow-depth: var(--_shadow-depth-light);

  --_transition-motion-reduce: ;
  --_transition-motion-ok:
    box-shadow 145ms ease,
    outline-offset 145ms ease
  ;
  --_transition: var(--_transition-motion-reduce);
}
```

{% endDetails %}

{% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/0SBiYWtJkz5JvF78HFPI.png", alt="Default buttons show in light and dark theme side by side.", width="800", height="188" %}

Dark theme adaptations

The value of the `-light` and `-dark` static props pattern becomes clear when the dark theme props are set:

```
@media (prefers-color-scheme: dark) {
  :where(
    button,
    input[type="button"],
    input[type="submit"],
    input[type="reset"],
    input[type="file"]
  ),
  :where(input[type="file"])::file-selector-button {
    --_bg: var(--_bg-dark);
    --_text: var(--_text-dark);
    --_border: var(--_border-dark);
    --_accent: var(--_accent-dark);
    --_highlight: var(--_highlight-dark);
    --_input-well: var(--_input-well-dark);
```

```
    --_ink-shadow: var(--_ink-shadow-dark);
    --_shadow-depth: var(--_shadow-depth-dark);
    --_shadow-color: var(--_shadow-color-dark);
    --_shadow-strength: var(--_shadow-strength-dark);
  }
}
```

Not only does this read well, but consumers of these custom buttons can use the bare props with confidence that they'll adapt appropriately to user preferences.

Reduced motion adaptations

If motion is OK with this visiting user, assign `--_transition` to `var(--_transition-motion-ok)`:

```
@media (prefers-reduced-motion: no-preference) {
  :where(
    button,
    input[type="button"],
    input[type="submit"],
    input[type="reset"],
    input[type="file"]
  ),
  :where(input[type="file"])::file-selector-button {
    --_transition: var(--_transition-motion-ok);
  }
}
```

A few shared styles

Buttons and inputs need to have their fonts set to `inherit` so they match the rest of the page fonts; otherwise they'll be styled by the browser. This also applies to `letter-spacing`. Setting `line-height` to `1.5` sets the letter box size to give the text some space above and below:

```
:where(
  button,
  input[type="button"],
  input[type="submit"],
  input[type="reset"],
  input[type="file"]
),
:where(input[type="file"])::file-selector-button {
```

```
  /* …CSS variables */

  font: inherit;
  letter-spacing: inherit;
  line-height: 1.5;
  border-radius: var(--_border-radius);
}
```

{% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/uEnOeiAMlwHeKNTAwAIs.png", alt="Screenshot showing buttons after the preceding styles have been applied.", width="800", height="190" %}

Styling buttons

Selector adjustment

The selector `input[type="file"]` is not the button part of the input, the pseudo-element `::file-selector-button` is, so I've removed `input[type="file"]` from the list:

```
:where(
  button,
  input[type="button"],
  input[type="submit"],
  input[type="reset"],
  input[type="file"]
),
:where(input[type="file"])::file-selector-button {

}
```

Cursor and touch adjustments

First I style the cursor to the `pointer` style, which helps the button indicate to mouse users that it's interactive. Then I add `touch-action: manipulation` to make clicks not need to wait and observe a potential double click, making the buttons feel faster:

```
:where(
  button,
  input[type="button"],
  input[type="submit"],
```

```
  input[type="reset"]
),
:where(input[type="file"])::file-selector-button {
  cursor: pointer;
  touch-action: manipulation;
}
```

{% Aside %} `touch-action: manipulation` isn't required if the HTML document has `<meta name="viewport" content="width=device-width">`. Learn more on Chrome Developers here. {% endAside %}

Colors and borders

Next I customize the font size, background, text, and border colors, using some of the adaptive custom properties established earlier:

```
:where(
  button,
  input[type="button"],
  input[type="submit"],
  input[type="reset"]
),
:where(input[type="file"])::file-selector-button {
  …

  font-size: var(--_size, 1rem);
  font-weight: 700;
  background: var(--_bg);
  color: var(--_text);
  border: 2px solid var(--_border);
}
```

{% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/SO94x6BLMRdRKYOJs4VF.png", alt="Screenshot showing buttons after the preceding styles have been applied.", width="800", height="182" %}

Shadows

The buttons have some great techniques applied. The `text-shadow` is adaptive to light and dark, creating a pleasing subtle appearance of the button text sitting nicely on top of the background. For the `box-shadow`, three shadows are assigned. The first, `--_shadow-2`, is a regular box shadow. The second shadow is a trick to the eye that makes the button appear to be beveled up a little bit.

The last shadow is for the hover highlight, initially at a size of 0, but it will be given a size later and transitioned so it appears to grow from the button.

```
:where(
  button,
  input[type="button"],
  input[type="submit"],
  input[type="reset"]
),
:where(input[type="file"])::file-selector-button {
  …

  box-shadow:
    var(--_shadow-2),
    var(--_shadow-depth),
    0 0 0 var(--_highlight-size) var(--_highlight)
  ;
  text-shadow: var(--_ink-shadow);
}
```

{% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/vVV9KIouLqJWM8Ya6UUf.png", alt="Screenshot showing buttons after the preceding styles have been applied.", width="800", height="173" %}

Layout

I gave the button a flexbox layout, specifically an `inline-flex` layout that will fit its content. I then center the text, and vertically and horizontally align children to the center. This will help icons and other button elements to align properly.

```
:where(
  button,
  input[type="button"],
  input[type="submit"],
  input[type="reset"]
),
:where(input[type="file"])::file-selector-button {
  …

  display: inline-flex;
  justify-content: center;
```

```
  align-items: center;
  text-align: center;
}
```

{% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/X9C3JQdj8uPWJwsTYoYe.png", alt="Screenshot showing buttons after the preceding styles have been applied.", width="800", height="176" %}

Spacing

For button spacing, I used `gap` to keep siblings from touching and logical properties for padding so button spacing works for all text layouts.

```
:where(
  button,
  input[type="button"],
  input[type="submit"],
  input[type="reset"]
),
:where(input[type="file"])::file-selector-button {
  …

  gap: 1ch;
  padding-block: var(--_padding-block);
  padding-inline: var(--_padding-inline);
}
```

{% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/aS1xe07aOnpyCeJjVbxz.png", alt="Screenshot showing buttons after the preceding styles have been applied.", width="800", height="176" %}

Touch and mouse UX

This next section is mostly for touch users on mobile devices. The first property, `user-select`, is for all users; it prevents text highlighting the button text. This is mostly noticeable on touch devices when a button is tapped and held and the operating system highlights the text of the button.

I've generally found this is not the user experience with buttons in built-in apps, so I disable it by setting `user-select` to none. Tap highlight colors (`-webkit-tap-highlight-color`) and operating system context menus (`-webkit-touch-callout`) are other very web-centric button features that aren't aligned with general button user expectations, so I remove them as well.

```
:where(
  button,
  input[type="button"],
  input[type="submit"],
  input[type="reset"]
),
:where(input[type="file"])::file-selector-button {
  …

  user-select: none;
  -webkit-tap-highlight-color: transparent;
  -webkit-touch-callout: none;
}
```

Transitions

The adaptive `--_transition` variable is assigned to the transition property:

```
:where(
  button,
  input[type="button"],
  input[type="submit"],
  input[type="reset"]
),
:where(input[type="file"])::file-selector-button {
  …

  transition: var(--_transition);
}
```

Upon hover, while the user is not actively pressing, adjust the shadow highlight size to give it a nice focus appearance that appears to grow from within the button:

```
:where(
  button,
  input[type="button"],
  input[type="submit"],
  input[type="reset"]
):where(:not(:active):hover) {
  --_highlight-size: .5rem;
```

```
}
```

{% Video src="video/vS06HQ1YTsbMKSFTIPl2iogUQP73/3O4JtOXT7JTrg1HqfQcR.mp4", autoplay="true", loop="true", muted="true", class="adjusted-aspect-ratio" %}

{% Aside %} The `::file-selector-button` is omitted from the selector because the effect is not able to be seen on it. {% endAside %}

Upon focus, increase the focus outline offset from the button, also giving it a nice focus appearance that appears to grow from within the button:

```
:where(button, input):where(:not(:active)):focus-visible {
  outline-offset: 5px;
}
```

{% Video src="video/vS06HQ1YTsbMKSFTIPl2iogUQP73/1mYWan5gC5tUjPdcmve3.mp4", autoplay="true", loop="true", muted="true", class="adjusted-aspect-ratio" %}

Icons

For handling icons, the selector has an added `:where()` selector for direct SVG children or elements with the custom attribute `data-icon`. The icon size is set with the custom property using inline and block logical properties. Stroke color is set, as well as a `drop-shadow` to match the `text-shadow`. `flex-shrink` is set to `0` so the icon is never squished. Lastly, I select lined icons and I assign those styles here with `fill: none` and `round` line caps and line joins:

```
:where(
  button,
  input[type="button"],
  input[type="submit"],
  input[type="reset"]
) > :where(svg, [data-icon]) {
  block-size: var(--_icon-size);
  inline-size: var(--_icon-size);
  stroke: var(--_icon-color);
  filter: drop-shadow(var(--_ink-shadow));

  flex-shrink: 0;
  fill: none;
  stroke-linecap: round;
  stroke-linejoin: round;
}
```

{% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/DOIaznn8R8QS51aWhpjR.png", alt="Screenshot showing buttons after the preceding styles have been applied.", width="800", height="178" %}

Customizing submit buttons

I wanted submit buttons to have a slightly promoted appearance, and I achieved this by making the text color of the buttons the accent color:

```
:where(
  [type="submit"],
  form button:not([type],[disabled])
) {
  --_text: var(--_accent);
}
```

{% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/D7BQA4GSeFShH0sJTopK.png", alt="Screenshot showing buttons after the preceding styles have been applied.", width="800", height="180" %}

Customize reset buttons

I wanted reset buttons to have some built-in warning signs to alert users of their potentially destructive behavior. I also chose to style the light theme button with more red accents than the dark theme. The customization is done by changing the appropriate light or dark underlying color, and the button will update the style:

```
:where([type="reset"]) {
  --_border-light: hsl(0 100% 83%);
  --_highlight-light: hsl(0 100% 89% / 20%);
  --_text-light: hsl(0 80% 50%);
  --_text-dark: hsl(0 100% 89%);
}
```

I also thought it'd be nice for the focus outline color to match the accent of red. The text color adapts a dark red to a light red. I make the outline color match this with the keyword `currentColor`:

```
:where([type="reset"]):focus-visible {
  outline-color: currentColor;
}
```

{% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/8llBDdQhUSKaMsaqKl7t.png", alt="Screenshot showing buttons after the preceding styles have been applied.", width="800",

height="179" %}

Customize disabled buttons

It's all too common for disabled buttons to have poor color contrast during the attempt to subdue the disabled button so it appears less active. I tested each color set and made sure they passed, nudging the HSL lightness value until the score passed in DevTools or VisBug.

```
:where(
  button,
  input[type="button"],
  input[type="submit"],
  input[type="reset"]
)[disabled] {
  --_bg: none;
  --_text-light: hsl(210 7% 40%);
  --_text-dark: hsl(210 11% 71%);

  cursor: not-allowed;
  box-shadow: var(--_shadow-1);
}
```

{% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/9hEOeFb5XrVV86iJATEq.png", alt="Screenshot showing buttons after the preceding styles have been applied.", width="800", height="176" %}

Customizing file input buttons

The file input button is a container for a span and a button. CSS is able to style the input container a little bit, as well as the nested button, but not the span. The container is given `max-inline-size` so it won't grow larger than it needs to, while `inline-size: 100%` will allow itself to shrink and fit containers smaller than it is. The background color is set to an adaptive color that is darker than other surfaces, so it looks behind the file selector button.

```
:where(input[type="file"]) {
  inline-size: 100%;
  max-inline-size: max-content;
  background-color: var(--_input-well);
}
```

{% Video src="video/vS06HQ1YTsbMKSFTIPl2iogUQP73/55vHgwPtOnXQlTY3Y9hz.mp4", autoplay="true", loop="true", muted="true" %}

The file selector button and input type buttons are specifically given `appearance: none` to remove any browser-provided styles that weren't overwritten by the other button styles.

```
:where(input[type="button"]),
:where(input[type="file"])::file-selector-button {
  appearance: none;
}
```

Lastly, margin is added to the `inline-end` of the button to push the span text away from the button, creating some space.

```
:where(input[type="file"])::file-selector-button {
  margin-inline-end: var(--_padding-inline);
}
```

{% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/KKy2Is3NsFjXAbltjrTW.png", alt="Screenshot showing buttons after the preceding styles have been applied.", width="800", height="180" %}

Special dark theme exceptions

I gave the primary action buttons a darker background for higher contrasting text, giving them a slightly more promoted appearance.

```
@media (prefers-color-scheme: dark) {
  :where(
    [type="submit"],
    [type="reset"],
    [disabled],
    form button:not([type="button"])
  ) {
    --_bg: var(--_input-well);
  }
}
```

{% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/DyIfTEeirgpu1LvJFUCc.png", alt="Screenshot showing buttons after the preceding styles have been applied.", width="800", height="177" %}

Creating variants

For fun, and because it's practical, I chose to show how to create a few variants. One variant is very vibrant, similar to how primary buttons often look. Another variant is large. The last variant has a gradient-filled icon.

Vibrant button

To achieve this button style, I overwrote the base props directly with blue colors. While this was quick and easy, it removes the adaptive props and looks the same in both light and dark themes.

```
.btn-custom {
  --_bg: linear-gradient(hsl(228 94% 67%), hsl(228 81% 59%));
  --_border: hsl(228 89% 63%);
  --_text: hsl(228 89% 100%);
  --_ink-shadow: 0 1px 0 hsl(228 57% 50%);
  --_highlight: hsl(228 94% 67% / 20%);
}
```

{% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/2XDwuP22sO0HFzKrDzYA.png", alt="Custom button is shown in light and dark. It is very vibrant blue like typical primary action buttons tend to be.", width="800", height="177" %}

{% Aside %} If I wanted to build an adaptive primary action button, I would overwrite `--_bg-light` and `--_bg-dark` instead of `--_bg`. Overwriting these variables allows the base property to swap between the new defined light and dark props. {% endAside %}

Large button

This style of button is achieved by modifying the `--_size` custom property. Padding and other space elements are relative to this size, scaling along proportionally with the new size.

```
.btn-large {
  --_size: 1.5rem;
}
```

{% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/jw8Dl101A8qDRWzb2ZcN.png", alt="Large button is shown next to the custom button, about 150 times larger.", width="800", height="174" %}

Icon button

This icon effect doesn't have anything to do with our button styles, but it does show how to achieve it with just a few CSS properties, and how well the button handles icons that aren't inline SVG.

```
[data-icon="cloud"] {
  --icon-cloud: url("https://api.iconify.design/mdi:apple-icloud.svg") center / contain no-repeat;

  -webkit-mask: var(--icon-cloud);
  mask: var(--icon-cloud);
  background: linear-gradient(to bottom, var(--_accent-dark), var(--_accent-light));
}
```

{% Img src="image/vS06HQ1YTsbMKSFTIPl2iogUQP73/Ti4EpUHGSLoCksVQHwmk.png", alt="A button with an icon is shown in light and dark themes.", width="800", height="173" %}

Conclusion

Now that you know how I did it, how would you? □□

Let's diversify our approaches and learn all the ways to build on the web.

Create a demo, tweet me links, and I'll add it to the community remixes section below!

Community remixes

Nothing to see here yet.

Resources

- Source code on Github

□□□ #2
□ □□□ □□□ 2026-10-31 21:40:39 UTC
□ □□□ □□□ 2026-10-31 21:40:40 UTC