Guidelines for CSS classes using BEM

- The guidelines
  - Syntax
  - Examples of syntax
- See also

When writing CSS classes for Discourse components, themes, or plugins, we follow a modified variant of Block Element Modifier (BEM). Following these guidelines will make it much less likely for CSS conflicts to occur, since BEM helps a lot with specificity. Themes and components have an easier time of overriding Discourse core styles, and the more descriptive class names make it easy at a glance to see where CSS classes should be applied in core.

The guidelines

This is what the Discourse engineering and design teams have aligned on using. It's mainly BEM, with some influence from SMACSS and others.

Syntax

It is generally a good idea to make a distinct block level CSS class per reusable Ember component, which you can then attach element CSS classes and modifiers to.

- You use default BEM for block and element in the format `.block__element` for example `.header__item` or `.admin-new-feature-item__screenshot`.
  - A block is a standalone component. Blocks can be used within blocks.
  - An element is a part of a block that can not be used outside of that block's context, for example an inner element.
  - A modifier you can use mainly for changing the appearance, if different than the default. This can be created standalone, because this can sometimes be re-used and

because this reduces pure BEM verbosity by a lot.
    - For example, `d-button` is the block which has some default styling, and `--cancel` is a modifier to make it look different
- These special prefixes can be used to signify that the piece of UI in question is currently styled a certain way because of a state or condition:
    - `is-foo`, for example `is-open` (for modals)
    - `has-foo`, for example `has-errors` (for forms)

Examples of syntax

Visually nesting is recommended because this keeps all relevant elements attached to the block and can be easily collapsed.

```
.block {
  // block styling

  &__element {
   //element styling

   &.--modifier { // direct modifier on element }
   .--modifier & { // indirect modifier on parent element}
  }
}
```

For modifiers, they can either be applied directly to an element like so:

```
<button class="d-button --modifier"></button>
```

Or in an alternate use case, imagine we want to style all of these elements inside the main block, when there is an error:

```
<div class="block --error">
   <input class="block__input"/>
   <label class="block__label">
   <p class="block__text">…</p>
</div>
```

Instead of placing a modifier on each element separately, we can just place it on the block level and use it indirectly via the syntax with `&` at the end. It makes little difference in the CSS file, but it keeps the DOM cleaner by not repeating modifiers.

A great real world example of our CSS classes in use in Discourse is within the chat plugin, in the loading skeleton component.

See also

- Designing for Different Devices — adapting styles to viewport size, touch vs. hover, and other device characteristics.

□□□ #2
□ □□□ □□□ 2026-10-31 13:28:39 UTC
□ □□□ □□□ 2026-10-31 13:28:39 UTC