# Contributing to Shoelace

Shoelace is an open source project, meaning everyone can use it and contribute to its development. Please take a moment to review these guidelines to make the contribution process as easy as possible for both you and the project's maintainers.

## Using the Issue Tracker

The [issue tracker](https://github.com/shoelace-style/shoelace/issues) is for bug reports, feature requests, and pull requests.

- Please \*\*do not\*\* use the issue tracker for personal support requests. Please [ask your question](https://stackoverflow.com/questions/ask) on StackOverflow instead.

- Please \*\*do not\*\* derail, hijack, or troll issues. Keep the discussion on topic and be respectful of others.

- Please \*\*do not\*\* post comments with "+1" or "?". Use [reactions](https://github.blog/2016-03-10-add-reactions-to-pull-requests-issues-and-comments/) instead.

- Please \*\*do\*\* use the issue tracker for feature requests, bug reports, and pull requests.

Issues that do not follow these guidelines are subject to closure. There simply aren't enough resources for the author and contributors to troubleshoot personal support requests.

### Feature Requests

Feature requests can be added using the issue tracker.

- Please \*\*do\*\* search for an existing request before suggesting a new feature.

- Please \*\*do\*\* use the "?" reaction to vote for a feature.

- Please \*\*do\*\* share substantial use cases and perspective that support new features if they haven't already been mentioned.

- Please \*\*do not\*\* bump, spam, or ping contributors to prioritize your own feature.

If you would like your feature prioritized, please consider [sponsoring the project](https://github.com/sponsors/claviska).

### Bug Reports

A bug is \_a demonstrable problem\_ caused by code in the library. Bug reports are an important contribution to the quality of the project. When submitting a bug report, there are a few steps you can take to make sure your issues gets attention quickly.

- Please \*\*do not\*\* paste in large blocks of irrelevant code

- Please \*\*do\*\* search for an existing issue before creating a new one

- Please \*\*do\*\* explain the bug clearly

- Please \*\*do\*\* provide a minimal test case that demonstrates the bug (e.g. [jsfiddle.net](https://jsfiddle.net/) or [CodePen](https://codepen.io/))

- Please \*\*do\*\* provide additional information, when necessary, to replicate the bug

\*\*A minimal test case is critical to a successful bug report.\*\* It demonstrates that the bug exists in the library and not in surrounding code. Contributors should be able to understand the bug without studying your code, otherwise they'll probably move on to another bug.

If you would like your bug prioritized, please consider [sponsoring the project](https://github.com/sponsors/claviska).

### Pull Requests

To keep the project on track, please consider the following guidelines before submitting a PR.

- Please \*\*do not\*\* submit a PR without opening an issue first, especially for non-trivial changes. This may prevent you from doing work that won't be accepted for various reasons (e.g. someone is already working on it, it's not a good fit for the project, it needs additional planning, etc.)

- Please \*\*do\*\* make sure your PR clearly defines what you're changing. Even if you feel your changes are obvious, please explain them so other contributors can more easily review your works. PRs without detailed descriptions are subject to closure pending more details.

- Please \*\*do not\*\* edit anything in `dist/`. These files are generated automatically, so you need to edit the source files instead.

The author reserves the right to reject any PR that's outside the scope of the project or doesn't meet code quality standards.

### Branches

`current` - This branch reflects the latest release and powers [shoelace.style](https://shoelace.style/).

`next` - This branch is where pull requests will land. It reflects what's coming in the \_next\_ release, which can be previewed at [next.shoelace.style](https://next.shoelace.style/).

## Conventions

This section aims to describe some of the design decisions and code conventions that support the project. All contributors are expected to follow conventions and best practices, even those not explicitly defined in this document. When in doubt, refer to existing source code and execute your best judgment.

In order to keep the project in a maintainable state, code that doesn't follow conventions and best practices will need to be refactored before it will be accepted. This isn't a knock on your code or your style — it's something the author considers necessary to operate a successful open source project.

### Semantic Versioning

Shoelace follows [Semantic Versioning](https://semver.org/). Breaking changes in components with the "Stable" badge will not be accepted until the next major version. As such, all contributions must consider the project's roadmap and take this into consideration. Features that are deemed no longer necessary will be deprecated but not removed.

Components with the "Experimental" badge should not be used in production. They are made available as release candidates for development and testing purposes. As such, changes to experimental components will not be subject to semantic versioning.

### Code Formatting & Linting

The majority of code formatting is handled automatically by [Prettier](https://prettier.io/) at build time. However, for the best experience, you should [install it in your editor](https://prettier.io/docs/en/editors.html) and enable format on save.

Linting is run automatically at build time. By design, the project will not build if the linter is unhappy. Contributors are strongly encouraged to install an ESLint plugin for your editor for the best developer experience.

### BEM Class Names

All components use a [shadow DOM](https://developer.mozilla.org/en-US/docs/Web/Web\_Components/Using\_shadow\_DOM), so styles are completely encapsulated from the rest of the document. As a result, class names \_inside\_ a component won't conflict with class names \_outside\_ the component, so we're free to name them whatever we want.

Internally, each component uses the [BEM methodology](http://getbem.com/) for its class names. There is no technical requirement to do this — it's purely the preference of the author. However, all contributions are expected to follow this practice for the sake of consistency.

## Developer Tips

### Documentation is the key to success

Nobody likes writing documentation, but not having good docs is frustrating to users and makes the project less appealing. Fortunately, writing documentation for Shoelace is really easy!

A lot of the documentation is generated automatically based on JSDoc comments in each component's source code. If you have ESLint enabled, your editor will tell you when a comment is missing. This may feel a bit naggy, but it's a very good thing. Every property, method, event, etc. is documented this way. By adding comments as you code, the docs are more easily kept up to date.

The documentation pages are served up by [Docsify](https://docsify.now.sh/). Check out `docs/\_sidebar.md` and `docs/components/` to get an idea of how pages are created and formatted. If you're creating a new component, it may help to use an existing component's markdown file as a template.

If you need help with documentation, feel free to reach out!

### Choose composability when possible

When designed right, web components are highly composable, meaning you can easily reuse them in other components. This reduces the overall size of the library, expedites feature development, and maintains a consistent user experience throughout.

The select component, for example, makes use of the dropdown, input, and menu components. Because it's offloading most of its functionality and styles to these lower-level components, the select component remains very lightweight and its appearance and behavior is consistent with other form controls and menus.

### When to use a standard property vs. a CSS custom property

When designing a component's API, standard properties ("props") are generally used to change the \_behavior\_ of a component, whereas CSS custom properties ("CSS variables") are used to change the \_appearance\_ of a component. Remember that props can't respond to media queries, but CSS variables can.

There are some exceptions to this (e.g. when it significantly improves developer experience), but a good rule of thumbs is "will this need to change based on screen size?" If so, you probably want to use a CSS variable.

### Boolean Props

Boolean props should \_always\_ default to `false`, otherwise there's no way for the user to unset it without JavaScript. To keep the API as friendly and consistent as possible, use a name like `noHeader` with a default value of `false` instead of `header` with a default value of `true`.

### A Note About Sass

The Shoelace \_source\_ is developed using Sass for the convenience of nested selectors, imports, and tedious things such as color palette generation. By design, Sass variables, color functions, and other preprocessor-specific feaures are not used in the source and will not be accepted in a PR.

Consumers of the library should never need to worry about preprocessing the library.

### Positioning Popovers

Shoelace uses an internal popover utility for dropdowns, tooltips, etc. This is a light abstraction of Popper.js designed to make positioning and transitioning things easy and consistent throughout the library. When possible, use this utility instead of relying on Popper directly. See `src/utilities/popover.ts` for details.

### Host and Base Elements

All components have a host element, which is a reference to the actual `<sl-\*>` elements. Make sure to always set the host element's `display` property to the appropriate value depending on your needs, as the default is `inline` per the HTML spec.

```css

:host {

display: inline-block;

}

```

Aside from `display`, avoid setting properties on the host element when possible. The reason for this is that styles applied to the host element are not encapsulated. Instead, create a base element that wraps the internals and style that instead. This convention also makes it easier to use BEM in components.

```css

.sl-example {

/\* This is the base element \*/

}

.sl-example--primary {

/\* Primary modifier \*/

}

```

To expose custom properties for a component, define them in the `:host` block and use the following syntax for comments so they appear in the generated docs.

```css

/\*\*

\* @param --color: The component's text color.

\* @param --background-color: The component's background color.

\*/

:host {

--color: white;

--background-color: tomato;

display: inline-block;

}

```