SkateJS

Effortless custom elements for modern view libraries.

Code HTML Result

```
// @jsx h

import { props, withComponent } from 'skatejs';
import withPreact from '@skatejs/renderer-preact';
import { h } from 'preact';

class WithPreact extends withComponent(withPreact()) {
  static get props() {
    return {
      name: props.string
    };
  }
  render({ name }) {
    return <span>Hello, {name}!</span>;
  }
}

customElements.define('with-preact', WithPreact);
```


Skate is a functional abstraction over the web component standards that:

- Produces cross-framework compatible components.
- Abstracts away common attribute / property semantics via `props`, such as attribute reflection and coercion.
- Adds several lifecycle callbacks for responding to prop updates, rendering and updating, as well as a way to manage internal component state.
- Provides a base set of mixins that hook into renderers such as @skatejs/renderer-preact .

Installing

Skate is on NPM:

```
npm install skatejs
```

The core principle of Skate is to provide abstractions for writing custom elements based on best-practices; things that aren't controversial. However, templating can be highly contentious. For this reason, Skate provides a hook to inject renderers for any view library. For example, if you wanted to write your custom elements with Preact, you'd install it like so:

```
npm install skatejs @skatejs/renderer-preact preact
```

There are renderers for many popular view libraries:

- LitHTML
- Preact
- React
- Or a custom renderer !

Usage

This is how you might write a web component using Skate and Preact:

```
// @jsx h

import { props, withComponent } from 'skatejs';
import withPreact from '@skatejs/renderer-preact';
import { h } from 'preact';

class WithPreact extends withComponent(withPreact()) {
  static get props() {
    return {
      name: props.string
    };
  }
  render({ name }) {
    return <span>Hello, {name}!</span>;
  }
}

customElements.define('with-preact', WithPreact);
```

Getting started

To get up and running with Skate, head over to the getting started guide .

Polyfills

Skate builds upon the Custom Elements and the Shadow DOM standards. Skate is capable of operating without the Shadow DOM — it just means you don't get any encapsulation of your component's HTML or styles. It also means that it's up to you to provide a way to project content (i.e. `<slot>`). It's highly recommended you use Shadow DOM whenever possible.

Though most modern browsers support these standards, some still need polyfills to implement missing or inconsistent behaviours for them.

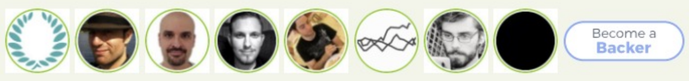
For more information on the polyfills, see the web components polyfill documentation .

Browser Support

Skate supports all evergreens and IE11, and is subject to the browser support matrix of the polyfills.

Backers

Support us with a monthly donation and help us continue our activities. Become a backer !



Sponsors

Become a sponsor and get your logo on our README on Github with a link to your site. Become a sponsor !