readme.md for @design.estate/dees-element

A powerful custom element base class that extends Lit's `LitElement` with integrated theming, responsive CSS utilities, RxJS-powered directives, and DOM tooling — so you can build web components that look great and stay reactive out of the box.

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

Install

```
npm install @design.estate/dees-element
# or
pnpm install @design.estate/dees-element
```

This package ships as ESM and is written in TypeScript. Make sure your project targets ES2022+ with a modern module resolution strategy (e.g. `NodeNext`).

Usage

Everything you need is exported from the main entry point:

```
import {
  DeesElement,
  customElement,
  property,
  state,
  html,
  css,
  cssManager,
  directives,
} from '@design.estate/dees-element';
```

□□ Creating a Custom Element

Extend `DeesElement` and apply the `@customElement` decorator:

```
import { DeesElement, customElement, html, css, cssManager } from '@design.estate/dees-element';

@customElement('my-button')
class MyButton extends DeesElement {
  static styles = [
    cssManager.defaultStyles,
    css`
      .btn {
        padding: 8px 16px;
        border-radius: 4px;
        background: ${cssManager.bdTheme('#0060df', '#3a8fff')};
        color: ${cssManager.bdTheme('#fff', '#fff')};
        border: none;
        cursor: pointer;
      }
    `,
  ];

  render() {
    return html`<button class="btn"><slot></slot></button>`;
  }
}
```

That single `bdTheme()` call generates a CSS variable that automatically flips between the bright and dark values when the user's theme changes — no manual toggling needed.

Theme Management with `cssManager`

The singleton `cssManager` is the central hub for theming and responsive layout:

Method	Purpose
`cssManager.defaultStyles`	Base styles for consistent element rendering
`cssManager.bdTheme(bright, dark)`	Returns a `CSSResult` that auto-switches between bright/dark values
`cssManager.cssForDesktop(css, this?)`	Breakpoint for desktop; pass `this` for component-scoped
`cssManager.cssForNotebook(css, this?)`	Breakpoint for notebook; pass `this` for component-scoped
`cssManager.cssForTablet(css, this?)`	Breakpoint for tablet; pass `this` for component-scoped
`cssManager.cssForPhablet(css, this?)`	Breakpoint for phablet; pass `this` for component-scoped
`cssManager.cssForPhone(css, this?)`	Breakpoint for phone; pass `this` for component-scoped
`cssManager.cssForConstraint({ maxWidth, minWidth })`	Custom viewport-level constraint (curried)
`cssManager.cssGridColumns(cols, gap)`	Generates CSS grid column widths

Example — responsive + themed styles:

```
@customElement('my-card')
class MyCard extends DeesElement {
  static styles = [
    cssManager.defaultStyles,
    css`
      :host {
        display: block;
        padding: 16px;
        background: ${cssManager.bdTheme('#ffffff', '#1e1e1e')};
        color: ${cssManager.bdTheme('#111', '#eee')};
        border-radius: 8px;
      }
    `,
    cssManager.cssForPhone(css`
      :host { padding: 8px; }
    `),
  ];
```

```
  render() {
    return html`<slot></slot>`;
  }
}
```

Container-Responsive Components

For components that need to respond to their **own width** (not the viewport), use the `@containerResponsive()` decorator and pass `this` as the second argument to `cssManager.cssFor*`:

```
import {
  DeesElement, customElement, html, css, cssManager,
  containerResponsive,
} from '@design.estate/dees-element';

@containerResponsive()
@customElement('my-stats-grid')
class MyStatsGrid extends DeesElement {
  static styles = [
    cssManager.defaultStyles,
    css`.grid { display: grid; grid-template-columns: repeat(3, 1fr); gap: 16px; }`,

    // Component-level: when THIS element is narrower than tablet width
    cssManager.cssForTablet(css`
      .grid { grid-template-columns: repeat(2, 1fr); }
    `, this),

    // Viewport-level: when the browser window is phone-sized
    cssManager.cssForPhone(css`
      .grid { grid-template-columns: 1fr; }
    `),

    // Component-level with custom width constraint
    this.cssForConstraint({ maxWidth: 500 })(css`
      .grid { gap: 8px; }
    `),
  ];
```

```
  render() {
    return html`<div class="grid"><slot></slot></div>`;
  }
}
```

How it works:

API	Scope	Generated CSS
`cssManager.cssForPhablet(css)`	Viewport	`@media` + `@container wccToolsViewport`
`cssManager.cssForPhablet(css, this)`	Component	`@container <tag-name>` only
`cssManager.cssForConstraint({maxWidth:800})(css)`	Viewport	`@media` + `@container wccToolsViewport`
`this.cssForConstraint({maxWidth:500})(css)`	Component	`@container <tag-name>` only
`@containerResponsive()`	Decorator	Sets `container-type: inline-size` + `container-name` on `:host`

The `@containerResponsive()` decorator is required for component-scoped queries — it establishes the CSS containment context on `:host`.

⚡ Reactive Properties & State

Use the standard Lit decorators, re-exported for convenience:

```
import { DeesElement, customElement, property, state, html } from '@design.estate/dees-element';

@customElement('my-counter')
class MyCounter extends DeesElement {
  @property({ type: String })
  accessor label = 'Count';

  @state()
  accessor count = 0;

  render() {
    return html`
      <button @click=${() => this.count++}>
        ${this.label}: ${this.count}
      </button>
```

```
    `;
  }
}
```

> **Note:** This library uses the TC39 standard decorators with the `accessor` keyword for decorated class properties.

Theme Change Callbacks

`DeesElement` tracks the current theme via the `goBright` property and exposes an optional `themeChanged` callback:

```
@customElement('theme-aware')
class ThemeAware extends DeesElement {
  protected themeChanged(goBright: boolean) {
    console.log(goBright ? 'Switched to bright' : 'Switched to dark');
  }

  render() {
    return html`<p>Current theme: ${this.goBright ? 'bright' : 'dark'}</p>`;
  }
}
```

Lifecycle Helpers

`DeesElement` adds lifecycle utilities on top of LitElement:

```
@customElement('my-widget')
class MyWidget extends DeesElement {
  constructor() {
    super();

    // Runs once after the element is connected to the DOM
    this.registerStartupFunction(async () => {
      console.log('Widget connected!');
    });
```

```
    // Runs when the element is disconnected — perfect for cleanup
    this.registerGarbageFunction(() => {
      console.log('Widget removed');
    });
  }

  render() {
    return html`<p>Hello World</p>`;
  }
}
```

Additionally, `this.elementDomReady` is a promise that resolves after `firstUpdated`, which is handy when you need to wait for the initial render:

```
await this.elementDomReady;
// The element's shadow DOM is now fully rendered
```

Directives

The `directives` namespace includes powerful template helpers, accessible via `directives.*`:

`resolve` — Render a Promise

```
import { html, directives } from '@design.estate/dees-element';

render() {
  return html`${directives.resolve(this.fetchData())}`;
}
```

`resolveExec` — Resolve a lazy async function

```
render() {
  return html`${directives.resolveExec(() => this.loadContent())}`;
}
```

`subscribe` — Render an RxJS Observable

```
import { html, directives } from '@design.estate/dees-element';
```

```
render() {
  return html`<span>${directives.subscribe(this.count$)}</span>`;
}
```

`subscribeWithTemplate` — Observable + template transform

```
render() {
  return html`
    ${directives.subscribeWithTemplate(
      this.items$,
      (items) => html`<ul>${items.map(i => html`<li>${i}</li>`)}</ul>`
    )}
  `;
}
```

Re-exported Lit directives

The directives namespace also re-exports these commonly used Lit directives:

- `until` — render a placeholder while a promise resolves
- `asyncAppend` — append values from an async iterable
- `keyed` — force re-creation of a template when a key changes
- `repeat` — efficiently render lists with identity tracking

□□ Full Export Reference

Export	Description
`DeesElement`	Base class for custom elements
`CssManager`	CSS/theme management class
`cssManager`	Singleton `CssManager` instance
`customElement`	Class decorator to register elements
`property`	Reactive property decorator
`state`	Internal state decorator
`query`, `queryAll`, `queryAsync`	Shadow DOM query decorators
`html`	Lit html template tag
`css`	Lit css template tag
`unsafeCSS`	Create `CSSResult` from a string

Export	Description
`unsafeHTML`	Render raw HTML in templates
`render`	Lit render function
`static` / `unsafeStatic`	Static html template helpers
`containerResponsive`	Decorator that adds CSS containment to `:host`
`domtools`	DOM tooling utilities
`directives`	All directives (resolve, subscribe, etc.)
`rxjs` (type)	RxJS type re-export

License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the LICENSE file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #6
Created 2026-03-28 10:49:10 UTC by foss.global Team
Updated 2026-03-28 12:14:34 UTC by foss.global Team