Custom Elements Everywhere

Making sure frameworks and custom elements can be BFFs

What's this?

Custom Elements are the lynchpin in the Web Components specifications. They give developers the ability to define their own HTML elements. When coupled with Shadow DOM, Custom Elements should be able to work in any application. But things don't always work seamlessly.

This project runs a suite of tests against each framework to identify interoperability issues, and highlight potential fixes already implemented in other frameworks. If frameworks agree on how they will communicate with Custom Elements, it makes developers' jobs easier; they can author their elements to meet these expectations.

Custom Elements and Shadow DOM don't come with a pre-defined set of best practices. The tests in this project are a best guess as to how things should work, but they're by no means final. This project is also about driving discussion and finding consensus, so don't be afraid to open a GitHub issue to discuss places where the tests could be improved. ✌️

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
Vue ^3.1.0	91%	16/16	8/14

Handling data

By default, Vue passes all data to Custom Elements as attributes. However, Vue also provides syntax to instruct its bindings to use properties instead. To bind to a Custom Element property use `:foo.prop="bar"` .

Handling events

Vue can listen to native DOM events dispatched from Custom Elements. Its declarative event bindings only support lowercase and kebab case events. To listen for any events named with capital letters you must write imperative code.

View the tests

Related Issues

Vue 3 does not support listening to custom events with capital letters

#5401 opened on Feb 10, 2022 by rictic

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
React 18.0.0	71%	16/16	0/14

Handling data

React passes all data to Custom Elements in the form of HTML attributes. For primitive data this is fine, but the system breaks down when passing rich data, like objects or arrays. In these instances you end up with stringified values like `some-attr="[object Object]"` which can't actually be used.

Handling events

Because React implements its own synthetic event system, it cannot listen for DOM events coming from Custom Elements without the use of a workaround. Developers will need to reference their Custom Elements using a `ref` and manually attach event listeners with `addEventListener`. This makes working with Custom Elements cumbersome.

View the tests

Related Issues

RFC: Plan for custom element attributes/properties in React 19
#11347 opened Oct 23 2017 by robdodson

Bypass synthetic event system for Web Component events
#7901 opened Oct 6, 2016 by staltz

Passing props to custom elements as properties instead of attributes
#8755 opened Jan 11, 2017 by joeldenning

Attributes and properties for Custom Components
#7249 opened Jul 12, 2016 by edoardocavazza

Boolean attributes on Web Components
#9230 opened Mar 21, 2017 by nickdima

Event Handler on React Component not invoked when React Component is rendered inside a Web Component
#9242 opened on Mar 23, 2017 by nilshartmann

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
React 0.0.0-experimental-79ed5e18f-20220217	100%	16/16	14/14

Experimental

The @experimental release of React features full support for Custom Elements, but the React core team has not yet reached a final decision on whether this will make it into React 18. Your feedback is important, and will be helpful for making this decision. More info at https://github.com/facebook/react/issues/11347#issuecomment-988970952

Handling data

React@experimental, as of February 2022, uses a runtime heuristic to determine if it should pass data to Custom Elements as either properties or attributes. If a property is already defined on the element instance, it will use properties, otherwise it will fallback to attributes.

Handling events

React@experimental, as of February 2022, will register an event listener on any custom element when binding a function to a property whose name begins with `on`. It supports lowercase, camelCase, kebab-case, CAPScase, and PascalCase events.

View the tests

Related Issues

RFC: Plan for custom element attributes/properties in React 19
#11347 opened Oct 23, 2017 by robdodson

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
Angular 13.3.0	100%	16/16	14/14

Handling data

Angular's default binding syntax will always set properties on an element. This works well for rich data, like objects and arrays, and also works well for primitive values so long as the Custom Element author has mapped any exposed attributes to corresponding properties.

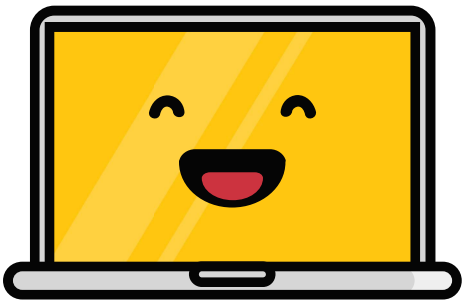
Angular also provides binding syntax specifically for setting an attribute, if a developer would prefer to communicate with an element that way.

Handling events

Angular components can listen to native DOM events dispatched from Custom Elements. It supports all styles of events (lowercase, camelCase, kebab-case, etc).

View the tests

Related Issues



Yay! No open issues!

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
AngularJS (1.x) 1.8.2	100%	16/16	14/14

Handling data

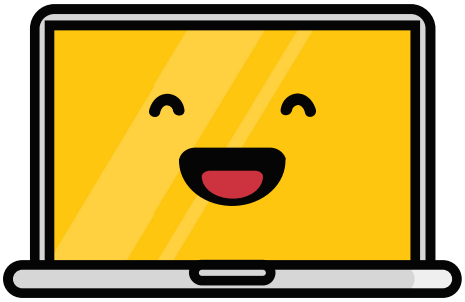
AngularJS can declaratively pass data to attributes using `ng-attr`, or to properties using `ng-prop`.

Handling events

AngularJS can declaritively listen to native DOM events by using `ng-on`. It supports all styles of events (lowercase, camelCase, kebab-case, etc) by prefixing uppercase characters with an underscore (`_`).

View the tests

Related Issues



Yay! No open issues!

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
Svelte 3.46.4	100%	16/16	14/14

Handling data

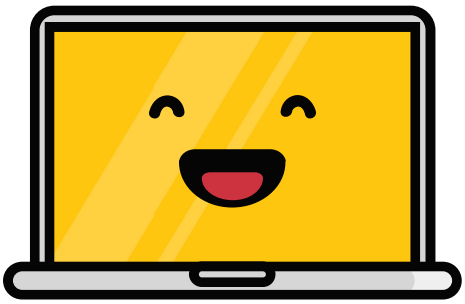
Svelte uses a heuristic to determine whether to pass data as properties or attributes — if the property is defined on the element instance, a property is used, otherwise it will fall back to attributes.

Handling events

Svelte can listen to native DOM events dispatched from Custom Elements. It supports all styles of events (lowercase, camelCase, kebab-case, etc).

View the tests

Related Issues



Yay! No open issues!

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
Preact ^10.0.1	100%	16/16	14/14

Handling data

Preact uses a runtime heuristic to determine if it should pass data to Custom Elements as either properties or attributes. If a property is already defined on the element instance, Preact will use properties, otherwise it will fallback to attributes. The exception to this rule is when it tries to pass rich data, like objects or arrays. In those instances it will always use a property.

Handling events

Preact can listen to native DOM events dispatched from Custom Elements. It uses a heuristic to convert JSX event binding syntax into event names. It supports lowercase, camelCase, kebab-case, CAPScase, and PascalCase events.

View the tests

Related Issues

check for typeof attribute value, if is string use setAttribute
#511 opened on Jan 17 by enapupe

Checking `prop in elem` fails with deferred custom element definitions
#678 opened on Apr 27 by treshugart

feat: Add native support for custom elements.
#715 opened on May 31 by treshugart

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
Polymer 3.4.1	91%	16/16	8/14

Handling data

Polymer will always attempt to pass data to an element using properties. To explicitly set an attribute, Polymer provides additional syntax in the form of the `$=` annotation.

Handling events

Polymer supports listening to DOM events using the `on-*` attribute syntax. It does *not* support arbitrarily capitalized event names (camelCase, CAPSCase, PascalCase, etc.). This is because Polymer reads the event name directly from the HTML attribute, and the HTML parser will always lowercase attribute names.

You can read more about this issue and why we test it in the FAQ.

View the tests

Related Issues

Polymer's declarative event listener names have issues with casing
#4888 opened Oct 17 by sorvell

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
Hyperapp 2.0.22	100%	16/16	14/14

Handling data

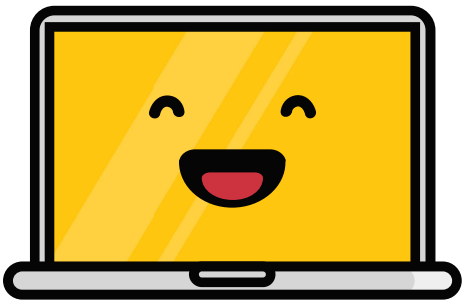
Hyperapp will pass data to an element as properties, as long as the property is defined on the element's prototype. Otherwise it will fallback to passing data as attributes.

Handling events

Hyperapp can listen to native DOM events dispatched from Custom Elements. It supports all styles of events (lowercase, camelCase, kebab-case, etc).

View the tests

Related Issues



Yay! No open issues!

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
Skate w/ Preact 5.2.4	91%	16/16	8/14

Handling data

Skate lets you use a variety of different rendering engines (Preact, React, lit-html). Most Skate apps these days use Preact, so Skate + Preact should pass data primarily using properties, and only fall back to attributes if a property is not defined.

Handling events

Skate's declarative event handling is defined by the rendering engine used. If you're using Skate + Preact then it will support events with lowercase and kebab-case names, but not camelCase, PascalCase, or CAPScase events (e.g. `'URLchanged'`).

View the tests

Related Issues

Allow mixed case events
#788 opened on August 2 by developit

check for typeof attribute value, if is string use setAttribute
#511 opened on Jan 17 by enapupe

Checking `prop in elem` fails with deferred custom element definitions
#678 opened on Apr 27 by treshugart

feat: Add native support for custom elements.
#715 opened on May 31 by treshugart

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
Riot.js 4.14.0	100%	16/16	14/14

Handling data

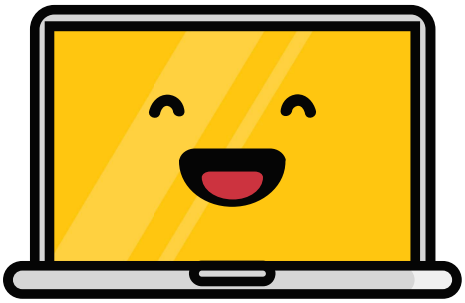
By default, Riot.js passes all primitive data (strings, numbers, booleans) to Custom Elements as attributes. It passes complex data (Objects, Arrays) to Custom Elements as properties.

Handling events

Riot.js can listen to native DOM events dispatched from Custom Elements. It supports all styles of events (lowercase, camelCase, kebab-case, etc).

View the tests

Related Issues



Yay! No open issues!

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
Mithril 2.0.4	100%	16/16	14/14

Handling data

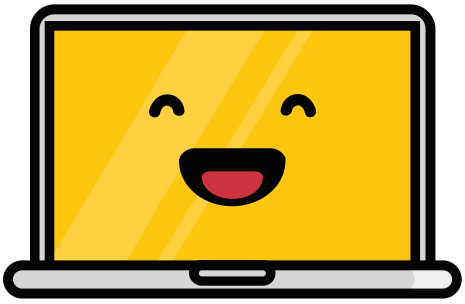
Mithril performs a `key in element` check to determine whether to assign values as properties or attributes: if the element or any of its prototypes have a property definition for the key in question, the value will be assigned as a property.

Handling events

Mithril can listen to native DOM events dispatched from Custom Elements. It supports all styles of events (lowercase, camelCase, kebab-case, etc).

View the tests

Related Issues



Yay! No open issues!

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
Omi 6.23.0	100%	16/16	14/14

Handling data

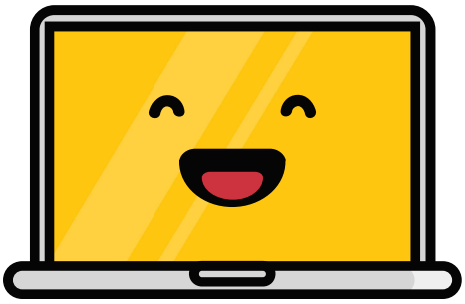
Omi uses a runtime heuristic to determine if it should pass data to Custom Elements as either properties or attributes. If a property is already defined on the element instance, Omi will use properties, otherwise it will fallback to attributes.

Handling events

Omi can listen to native DOM events dispatched from Custom Elements. It supports all styles of events (lowercase, camelCase, kebab-case, etc).

View the tests

Related Issues



Yay! No open issues!

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
Lit 2.2.1	100%	16/16	14/14

Handling data

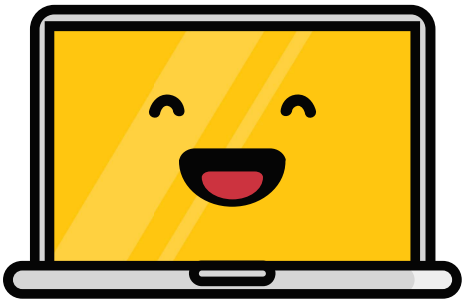
By default, Lit passes all data to Custom Elements as attributes. However, Lit also provides syntax to bind to properties instead. To bind to a Custom Element property, prefix the property name with a `.` as in `<input .value=${value}>`.

Handling events

Lit can listen to native DOM events dispatched from Custom Elements. It supports all styles of events (lowercase, camelCase, kebab-case, etc).

View the tests

Related Issues



Yay! No open issues!

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
Stencil 2.15.0	100%	16/16	14/14

Handling data

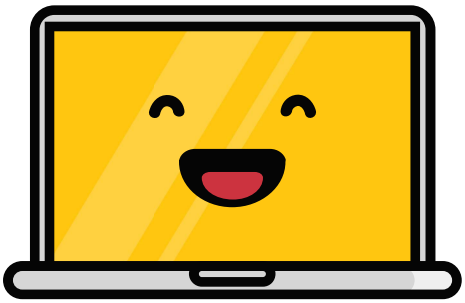
By default, Stencil passes all data to Custom Elements as properties.

Handling events

Stencil can listen to native DOM events dispatched from Custom Elements. It supports all styles of events (lowercase, camelCase, kebab-case, etc).

View the tests

Related Issues



Yay! No open issues!

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
hyperHTML 2.34.0	100%	16/16	14/14

Handling data

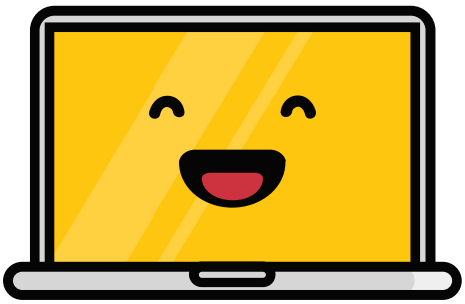
hyperHTML will pass data to an element as properties, as long as the property is defined on the element's prototype. Otherwise it will fallback to passing data as attributes.

Handling events

hyperHTML can listen to native DOM events dispatched from Custom Elements. It supports all styles of events (lowercase, camelCase, kebab-case, etc).

View the tests

Related Issues



Yay! No open issues!

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
hybrids 4.3.4	100%	16/16	14/14

Handling data

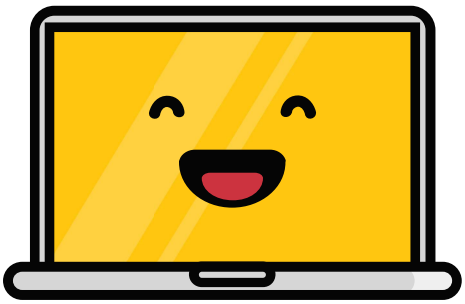
Hybrids will pass data to an element as properties, as long as the property is defined on the element's prototype. Otherwise it will fallback to passing data as attributes.

Handling events

Hybrids can listen to native DOM events dispatched from Custom Elements. It supports all styles of events (lowercase, camelCase, kebab-case, etc).

View the tests

Related Issues



Yay! No open issues!

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
DIO 9.1.1	91%	16/16	8/14

Handling data

DIO uses a runtime heuristic to determine if it should pass data to Custom Elements as either properties or attributes. If a property is already defined on the element instance, DIO will use properties, otherwise it will fallback to attributes. The exception to this rule is when it tries to pass rich data, like objects or arrays. In those instances it will always use a property.

Handling events

DIO can listen to native DOM events dispatched from Custom Elements. However, it uses a heuristic to convert JSX event binding syntax into event names, and always lowercases the events. For example `onFooUpdated={handleFoo}` tells DIO to listen for an event called `'fooupdated'`. This means DIO can support events with lowercase and kebab-case names, but not camelCase, PascalCase, or CAPScase events (e.g. `'URLchanged'`).

View the tests

Related Issues

Consider supporting mixed case events.

#44 opened on December 21 by robdodson

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
Surplus 0.5.3	100%	16/16	14/14

Handling data

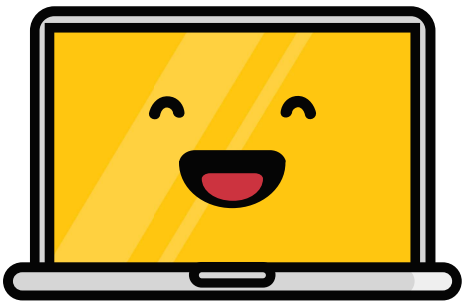
Surplus passes data to an element via properties unless the indicated field is known to be available only as an attribute (`aria-*` , some SVG attributes).

Handling events

By default, event handlers are registered in Surplus by setting the `node.on...` DOM properties: `<div onclick={...}></div>` . For custom events, which don't have such properties, Surplus uses surplus-mixin-on: `<div fn={on('my-custom-event', ...)}></div>` .

View the tests

Related Issues



Yay! No open issues!

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
Dojo 8.0.0	100%	16/16	14/14

Handling data

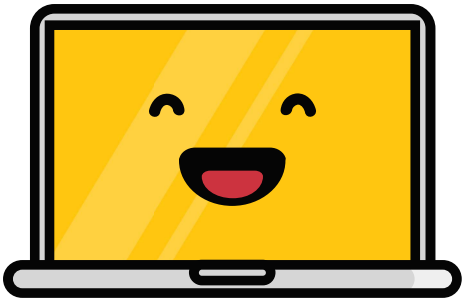
Dojo will pass data as attributes only when the data is a type of `string`, otherwise it is set as a property.

Handling events

Dojo can listen to native DOM events dispatched from Custom Elements. However the event names must be prefixed with `on`, so a Custom Event of `camelEvent` would be `oncamelEvent`. Other than that, Dojo supports all kinds of event names.

View the tests

Related Issues



Yay! No open issues!

LIBRARY	SCORE	BASIC TESTS	ADVANCED TESTS
Solid 1.3.13	100%	16/16	14/14

Handling data

Solid passes all non-JSX expression data as attributes. JSX expressions default to attributes unless they are booleans, applied to a custom element, or indicated with `prop:` namespace.

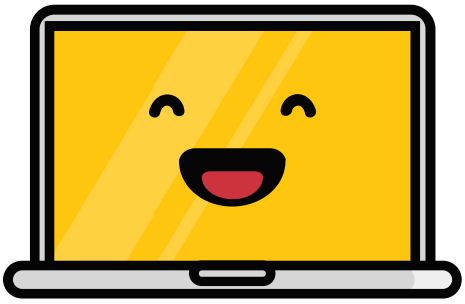
Handling events

By default, typical "on____" event handlers are registered in Solid using `<div onClick={...}></div>`. However, it always lowercases the event names and does automatic event delegation for input and mouse events.

For custom events with non-standard names, Solid uses its "on" binding: `<div on:my-custom-event={...}></div>`.

View the tests

Related Issues



Yay! No open issues!

Frequently Asked Questions

What are these custom element things?

Custom elements are a new web standard which let developers create their own HTML Elements. Because they're based on web standards, these elements should work on any page. This means, you can write a component, like a datepicker, and share it everywhere.

To learn more, check out the custom elements primer and best practices guide.

Are you testing that libraries let you author custom elements?

No. These tests just check that a library/framework supports the *usage* of custom elements. Essentially we're trying to answer the question: "If you're building an app in framework X, and you'd like to include a few custom elements on the page, are you going to have a bad time?"

The tests check that the library/framework will let you do things like display a custom element, bind data to it, pass in children, and listen for events.

Why is each test counted twice?

We run each test in Chrome and Firefox. These days they behave basically the same with web components but years ago when these tests were first written, Firefox didn't yet have native support for Web Components.

How are the libraries scored?

Each test has an associated weight, based on how critical it is. The final tally of pass/fails is combined with these weights to create a weighted average score.

How are basic and advanced tests different?

Basic tests cover things which are fundamental to a library/framework's ability to display a custom element. For example, can it display a custom element that contains shadow DOM? Can it handle setting attributes on the custom element? Can it listen for DOM events from the element? Failing any of these tests is a pretty critical issue.

Advanced tests cover more opinionated framework features. For example, does the framework provide declarative syntax for listening to events with

different casing styles (kebab-case, camelCase, etc). These are more like "nice to haves" that may improve the developer experience.

I thought the whole point of Polymer was to write custom elements. Why doesn't it get a 100%?

Polymer supports a non-standard feature called declarative event binding, which lets you use attributes to wire up event listeners. E.g. `<my-element on-foo="handleFoo">` . Because DOM events are just strings, there are no rules governing how they should be formatted or capitalized—"my-event" is just as reasonable as "myEvent" or "myevent". Even the web platform has a few examples of oddly cased events like `DOMContentLoaded` . Because Polymer's implementation of declarative event bindings relies on pulling the event name from the `on-*` attribute, and the HTML parser will *always lowercase attribute names*, it is unable to listen for events with capital letters in their names.

Since it is entirely possible to write a vanilla custom element that dispatches an event with a capital letter in its name, and because there is prior art in the platform that actually uses this technique for event names with acronyms ("DOM"), we feel it is important to test this.

Libraries like Preact also fail these capitalization tests, which points to a possible best practice of always making event names lowercase. This is the style most DOM events already use: `mousedown, popstate, beforeunload` , etc.

If a library is missing a feature, like declarative event or property bindings, does it automatically fail those tests?

Not necessarily. If a library omits a **non-standard** feature aimed at developer ergonomics, e.g. declarative event/property bindings, we would just omit those tests from the scoring process.

Why don't you have tests for _____ library?

We'd like to have as much test coverage as possible, but it's a fair bit of work building each test suite (especially because we are not experts in every library). If a framework or library is not represented it's just because we haven't had a chance to write tests for it yet.

I want to write some tests for _____ library. Do you accept pull requests?

Yes! In fact, >50% of the tests on this site are from external contributors.

If you'd like to contribute, please first open an issue saying you'd like to write some tests for a specific library/framework. This helps ensure that there's not more than one person writing tests for the same library.