Engineer

Web components

🖼 browser features for defining new, reusable HTML elements

What it is

Web components, also written Web Components with capitals, is the usual name for a group of browser features that let authors define their own HTML elements. A page then uses such an element like any other tag. The standards speak of the parts by their own names: custom elements and the `template` and `slot` elements in the HTML Standard, and shadow trees in the DOM Standard. The collective name is the title of a W3C Working Group Note of 24 July 2014, *Introduction to Web Components*, which the W3C has since discontinued and marked as a document that should not be referenced.

A custom element is defined by a JavaScript class that extends `HTMLElement` and is registered under a name with `customElements.define()`. The definition tells the HTML parser how to construct the element and how it should react to changes, and from then on the element can be created from markup, with `document.createElement()` or with `new`. Its name starts with a lower-case ASCII letter, contains no upper-case ASCII letters and must contain a hyphen. The hyphen keeps custom names apart from the language itself: the HTML Standard says that HTML, SVG and MathML will not gain any element whose name contains a hyphen.

There are two kinds. An autonomous custom element, such as the standard's own `<flag-icon country="nl">`, starts from a plain element with no built-in meaning. A customized built-in element extends an existing one: a class that extends `HTMLButtonElement`, registered with `{ extends: "button" }` and used as `<button is="plastic-button">`, keeps everything that makes a button a button, including focus behaviour, form submission and the `disabled` attribute.

A shadow tree is a separate tree of nodes attached to an element, which is called its host; the DOM Standard calls the host's ordinary tree the light tree. `attachShadow()` creates a shadow root in open or closed mode, and only in open mode does the host's `shadowRoot` property return it. A `slot` element in the shadow tree marks where the host's own children appear. It shows the nodes assigned to it, or its own contents when there are none, and a named slot takes the children whose `slot` attribute carries its name. The `template` element holds a fragment of HTML that is not rendered and that a script can clone. With a `shadowrootmode` attribute, a `template` represents a declarative shadow root, so a shadow tree can also be written straight into the markup.

Use the concept in a decision

Put "What are web components?" to work in a real decision. We can help relate the concept to the system you are building or buying. Interfaces that load quickly, work on a phone and behave the same in every browser your customers happen to use.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Directed full-stack GIS map development — full-stack mapping, from database to browser

Directed full-stack GIS map development Get in touch →

Why it matters

A made-up tag used to be of little use: the HTML Standard looks back on such elements as invalid and of little practical use. A custom element is a full member of the document. The parser constructs it, the DOM methods create it, and its class decides what happens when its attributes change or when it enters or leaves a page.

The definition does not have to arrive first. An element the parser has already created becomes a custom element once a definition for its name is registered, and the HTML Standard calls this upgrading. Until then the element is treated as an undefined element, much like a `span`, so the text and links inside it are shown as they are. The standard says in so many words that upgrades allow the content of a custom element to be progressively enhanced.

A custom element can also take part in forms and in the accessibility tree. A class that sets `static formAssociated = true` and calls `attachInternals()` gets an `ElementInternals` object, and `setFormValue()` on that object gives the element a value that its form submits like any other control's. The same object sets a default ARIA role and default ARIA states, which apply unless the page author

overrides them with the `role` attribute or the matching `aria-*` attributes, and it can hold custom states that style sheets match with the `:state()` pseudo-class.

For a team, the shadow tree is what makes a component self-contained: ordinary page selectors do not cross into its internal shadow tree, and the component decides which parts to expose for outside styling. Inherited values and custom properties can still enter, and slotted light-tree content remains subject to page selectors. Where several libraries share a page, the HTML Standard offers scoped registries, created with `new CustomElementRegistry()`, so that their elements can coexist without the libraries having to coordinate their names.

What it looks like in practice

A product card shows the parts working together. The page holds the content as ordinary markup, with a `slot` attribute on the element that should go into a named slot:

```
<product-card>
  <span slot="name">Oak stool</span>
  <a href="/shop/oak-stool/">See the details</a>
</product-card>
<script type="module" src="product-card.mjs"></script>
```

The module defines the element. The template is built once, and every card gets its own copy of it in an open shadow root:

```
const template = document.createElement("template");
template.innerHTML = `
  <h2 part="title"><slot name="name">Unnamed product</slot></h2>
  <slot></slot>
`;

class ProductCard extends HTMLElement {
  constructor() {
    super();
    const root = this.attachShadow({ mode: "open" });
    root.append(template.content.cloneNode(true));
  }
}
```

```
if (!customElements.get("product-card")) {
  customElements.define("product-card", ProductCard);
}
```

The page styles the title through the one part the component exposes:

```
product-card::part(title) {
  font-size: 1.25rem;
}
```

Before `product-card.mjs` has run, the card is an undefined element and the reader sees the product name and the link as plain inline content. When the module runs, `define()` registers the class and the card already in the page is upgraded. Its constructor attaches a shadow root and fills it with a clone of the template's contents. The `span` is assigned to the slot called `name` and appears inside the heading. The link has no `slot` attribute, so it goes to the unnamed default slot. A card without a name shows the slot's own contents, "Unnamed product", in its place. The `customElements.get()` check keeps a second copy of the module, arriving in another bundle, from making `define()` throw.

Shared components help only if people use them, and this site's portfolio has a case about that. The iOS 26 "Liquid Glass" design system starts from a UI in which every new screen reinvents its own buttons, badges and cards, each a little different, until every change means touching five bespoke versions. The answer described there is a canonical set of components that every screen composes from, among them Card, Label, Button, GlassIconButton and DirectoryGrid, with lint rules that reject a hand-rolled badge or chip and point to the canonical component. The page does not say whether those components are custom elements, but the discipline carries over: one definition per component, kept in force by tooling rather than by a reviewer's memory. The custom element registry enforces the first half by refusing a second definition for a name.

Where it goes wrong

A hyphenated name stands in for a role. "An autonomous custom element does not have any special meaning: it represents its children," says the HTML Standard. Its `taco-button` example shows the cost: to act as a button, the element needs a `tabindex`, the ARIA role `button`, an accessible name, `aria-disabled`, key handlers that turn Enter and Space into clicks, and styles for the disabled state.

Even then, a custom element cannot declare default focus behaviour, so `tabindex="0"` may not match how buttons behave on the platform. Where the element builds on an existing one, the standard says extending a built-in element reduces the work for both its author and the people using it. For an autonomous element, set its default role and states through `ElementInternals` rather than leaving them to every page that uses it.

The constructor does the work. The HTML Standard sets rules for custom element constructors: `super()` with no arguments comes first, the constructor must not look at the element's attributes or children, because an element created rather than upgraded has none, and the element must not gain attributes or children of its own. A constructor that breaks these rules can produce an element that the parser or the DOM methods cannot create. Use the constructor for initial state, default values, event listeners and the shadow root, and move fetching and rendering to `connectedCallback` .

`connectedCallback` **is expected to run once.** It can run more than once. By default, moving an element runs `disconnectedCallback` and then `connectedCallback` , so the element resets its state as if it had been removed and inserted again. Guard set-up that must happen only once with a flag, and undo in `disconnectedCallback` what `connectedCallback` started. If an element should keep its state when it is moved with `moveBefore()` , define `connectedMoveCallback` , whose presence alone replaces the default pair, even with an empty body.

A name is defined twice. `customElements.define()` throws a `NotSupportedError` when the name or the class is already registered, and a `SyntaxError` when the name is not valid, for example because it has no hyphen or contains a capital letter. Check with `customElements.get()` before defining, and give a library that must coexist with others a scoped registry of its own.

Closed mode is treated as protection. Closed mode hides the root from the host's `shadowRoot` property and hides internal nodes from an event's composed path as seen by listeners outside that closed shadow tree. CSS Shadow Parts states plainly that shadow trees are deliberately not a security boundary, only a convenience for page authors. Keep secrets out of components, and choose closed mode for its effect on outside code, not as a lock.

`attachShadow()` **is called where it cannot work.** The DOM Standard allows a shadow root only on elements with a valid custom element name and on a short list of built-in ones, among them `div` , `section` , `p` and `span` , but not

`button`. On a customized built-in button, the call throws a `NotSupportedError`. It also behaves differently on an element that already has a root: if that root is declarative and of the same mode, the call removes everything in it and returns it empty, and otherwise it throws. Check for an existing shadow root first, so that markup sent from the server in a declarative root is kept rather than wiped.

Page styles are expected to reach inside. CSS Shadow Parts calls the boundary a styling barrier, and notes that it also makes a component harder to style when the page wants to. Expose the elements a page may style with `part`, and style them from outside with `::part()`. A component nested inside another needs `exportparts` on the inner host to pass its parts on to the page. For colours and spacing that should follow the page, read custom properties set on the host, which reach the shadow tree by inheritance.

The content exists only after the script. A component that writes all its text from JavaScript shows readers none of it until the definition has loaded, and none at all if the script fails. Keep the content in the light tree and assign it to slots, as the product card does, so that upgrading enhances what is already there. Where content must wait, the HTML Standard's own example hides it until `customElements.whenDefined()` has resolved for every element matching `:not(:defined)`.

Further reading

- HTML Standard: Custom elements — naming rules, constructor requirements, upgrades, form-associated elements and `ElementInternals`, with worked examples.
- DOM Standard: Shadow tree — shadow roots and their hosts, slots and the nodes assigned to them, and the steps behind `attachShadow()`.
- HTML Standard: Scripting — the `template` element with its `shadowrootmode` attribute, and the `slot` element.
- CSS Shadow Parts — the W3C Working Draft of 16 December 2025 that defines `part`, `exportparts` and the `::part()` pseudo-element.

Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how “What are web components?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/web-components/