**Engineer**

# Web components

🖼 webbläsarfunktioner för nya, återanvändbara HTML-element

## Vad det är

Web components, även skrivet Web Components med versaler, är det vanliga namnet på en grupp webbläsarfunktioner som låter utvecklare definiera sina egna HTML-element. En sida använder sedan ett sådant element som vilken annan tagg som helst. Standarderna kallar delarna vid deras egna namn: custom elements och elementen `template` och `slot` i HTML-standarden, och shadow trees i DOM-standarden. Samlingsnamnet är titeln på *Introduction to Web Components*, en Working Group Note som W3C gav ut den 24 juli 2014 och sedan har avvecklat och märkt som ett dokument som inte bör hänvisas till.

Ett custom element definieras av en JavaScript-klass som utökar `HTMLElement` och registreras under ett namn med `customElements.define()` . Definitionen talar om för HTML-parsern hur elementet ska konstrueras och hur det ska reagera på ändringar, och från och med då kan elementet skapas från markup, med `document.createElement()` eller med `new` . Namnet börjar med en liten ASCII-bokstav, innehåller inga stora ASCII-bokstäver och måste innehålla ett bindestreck. Bindestrecket håller egna namn åtskilda från själva språket: HTML-standarden utlovar att inget element med bindestreck i namnet kommer att läggas till i HTML, SVG eller MathML framöver.

Det finns två sorter. Ett autonomt custom element, som standardens eget `<flag-icon country="nl">` , utgår från ett vanligt element utan inbyggd betydelse. Ett customized built-in element bygger ut ett befintligt element. Standardens exempel är en klass som utökar `HTMLButtonElement` , registreras med `{ extends: "button" }` och används som `<button is="plastic-button">` . Ett sådant element behåller allt som gör en knapp till en knapp, bland annat fokusbeteende, formulärinskickning och attributet `disabled` .

Ett shadow tree är ett separat träd av noder som är kopplat till ett element, som kallas dess värd; DOM-standarden kallar värdens vanliga träd för light tree. `attachShadow()` skapar en shadow root i läget open eller closed, och bara i läget open returnerar värdens egenskap `shadowRoot` den. Ett `slot` -element i värdens shadow tree markerar var värdens egna barn visas. Det visar de noder som har tilldelats det, eller sitt eget innehåll när inga har tilldelats, och en namngiven slot tar de barn vars `slot` -attribut bär dess namn. Elementet `template` rymmer ett HTML-fragment som inte renderas och som ett skript kan klona. Med attributet `shadowrootmode` representerar en `template` en deklarativ shadow root, så att ett shadow tree också kan skrivas direkt i markupen.

Använd begreppet i ett beslut

Använd ”Vad är web components?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Gränssnitt som laddar snabbt, fungerar i mobilen och beter sig likadant i alla webbläsare dina kunder råkar använda.

**Se våra tjänster** Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Styrde full stack-utvecklingen av GIS-kartor — kartutveckling i full stack, från databas till webbläsare

**Styrde full stack-utvecklingen av GIS-kartor** Hör av er →

## Varför det spelar roll

En påhittad tagg var tidigare till liten nytta: HTML-standarden konstaterar att sådana element historiskt har stridit mot standarden och inte varit särskilt funktionella. Ett custom element är en fullvärdig medlem av dokumentet. Parsern konstruerar det, DOM-metoderna skapar det, och dess klass bestämmer vad som händer när dess attribut ändras eller när det läggs in på eller tas bort från en sida.

Definitionen behöver inte komma först. Ett element som parsern redan har skapat blir ett custom element så snart en definition för dess namn registreras, och HTML-standarden kallar det en uppgradering. Fram till dess behandlas elementet som ett odefinierat element, ungefär som en `span` , så texten och länkarna i det visas som de är. Standarden säger rent ut att uppgraderingar gör det möjligt att förbättra innehållet i ett custom element progressivt.

Ett custom element kan också delta i formulär och i tillgänglighetsträdet. En klass som sätter `static formAssociated = true` och anropar `attachInternals()` får ett `ElementInternals` -objekt, och `setFormValue()` på det objektet ger elementet ett värde som formuläret skickar med precis som för vilken annan kontroll som

helst. Samma objekt sätter en standardroll och standardtillstånd i ARIA, som gäller om inte sidans författare åsidosätter dem med attributet `role` eller motsvarande `aria-*` -attribut, och det kan hålla egna tillstånd som stilmallar matchar med pseudoklassen `:state()` .

För ett team är vinsten med ett shadow tree att komponenten blir självständig: vanliga selektorer på sidan når inte in i dess interna shadow tree, och komponenten bestämmer vilka delar den ska exponera för styling utifrån. Ärvda värden och custom properties kan fortfarande nå in, och innehåll från light tree som visas i slots påverkas fortfarande av sidans selektorer. Där flera bibliotek delar en sida erbjuder HTML-standarden scoped registries, som skapas med `new CustomElementRegistry()` , så att deras element kan samexistera utan att biblioteken behöver samordna sina namn.

## Så ser det ut i praktiken

Ett produktkort visar delarna i samspel. Sidan har innehållet som vanlig markup, med ett `slot` -attribut på det element som ska in i en namngiven slot:

```
<product-card>
  <span slot="name">Ekpall</span>
  <a href="/butik/ekpall/">Se detaljerna</a>
</product-card>
<script type="module" src="product-card.mjs"></script>
```

Modulen definierar elementet. Mallen byggs en gång, och varje kort får en egen kopia av den i en öppen shadow root:

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

Sidan stylar rubriken genom den enda del som komponenten exponerar:

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

Innan `product-card.mjs` har körts är kortet ett odefinierat element, och läsaren ser produktnamnet och länken som vanligt innehåll i texten. När modulen körs registrerar `define()` klassen, och kortet som redan finns på sidan uppgraderas. Konstruktorn kopplar en shadow root till kortet och fyller den med en klon av mallens innehåll. `span` -elementet tilldelas den slot som heter `name` och visas inne i rubriken. Länken saknar `slot` -attribut, så den hamnar i den namnlösa standardsloten. Ett kort utan namn visar i stället slottens eget innehåll, ”Namnlös produkt”. Kontrollen med `customElements.get()` hindrar en extra kopia av modulen, som kommer med ett annat paket, från att få `define()` att kasta ett fel.

Gemensamma komponenter hjälper bara om folk använder dem, och webbplatsens portfölj har ett uppdrag som handlar om just det. Liquid Glass-designsystemet för iOS 26 utgår från ett gränssnitt där varje ny skärm uppfinner sina egna knappar, märken och kort, alla lite olika, tills varje ändring innebär att fem skräddarsydda versioner måste rättas. Svaret som sidan beskriver är en kanonisk uppsättning komponenter som alla skärmar byggs av, bland dem Card, Label, Button, GlassIconButton och DirectoryGrid, med lintregler som avvisar ett hemsnickrat märke eller chip och pekar mot den kanoniska komponenten. Sidan säger inte om komponenterna är custom elements, men disciplinen går att föra över: en enda definition per komponent, upprätthållen av verktyg och inte av granskarens minne. Registret för custom elements upprätthåller den första halvan genom att avvisa en ny definition för ett namn som redan används.

## Var det går fel

**Ett namn med bindestreck får ersätta en roll.** ”An autonomous custom element does not have any special meaning: it represents its children,” skriver HTML-standarden. Standardens exempel `taco-button` visar vad det kostar: för att bete

sig som en knapp behöver elementet ett `tabindex`, ARIA-rollen `button`, ett tillgängligt namn, `aria-disabled`, tangentbordshantering som gör Enter och mellanslag till klick, och stilar för det inaktiverade läget. Även då kan ett custom element inte ange ett standardbeteende för fokus, så `tabindex="0"` stämmer kanske inte med hur knappar beter sig på plattformen. Där elementet bygger på ett befintligt element säger standarden att ett customized built-in element blir lättare att utveckla, underhålla och använda. Ge ett autonomt element en standardroll och standardtillstånd via `ElementInternals` i stället för att lämna dem åt varje sida som använder det.

**Konstruktorn gör jobbet.** HTML-standarden ställer krav på konstruktorer i custom elements: `super()` utan argument kommer först, elementets attribut och barn får inte undersökas, eftersom ett element som har skapats och inte uppgraderats inte har några, och konstruktorn får inte ge elementet attribut eller barn. En konstruktor som bryter mot reglerna kan ge ett element som parsern eller DOM-metoderna inte kan skapa. Använd konstruktorn till starttillstånd, standardvärden, händelselyssnare och shadow root, och flytta hämtning och rendering till `connectedCallback`.

`connectedCallback` **förväntas köras en gång.** Den kan köras mer än en gång. Som standard leder en flytt av elementet till att `disconnectedCallback` och sedan `connectedCallback` körs, så att elementet nollställer sitt tillstånd som om det hade tagits bort och satts in igen. Skydda uppsättning som bara får ske en gång med en flagga, och städa i `disconnectedCallback` upp efter det som `connectedCallback` satte igång. Ska ett element behålla sitt tillstånd när det flyttas med `moveBefore()`, så definierar du `connectedMoveCallback`: att den finns räcker för att ersätta standardparet, även med en tom kropp.

**Ett namn definieras två gånger.** `customElements.define()` kastar ett `NotSupportedError` när namnet eller klassen redan är registrerad, och ett `SyntaxError` när namnet inte är giltigt, till exempel för att det saknar bindestreck eller innehåller en versal. Kontrollera med `customElements.get()` innan du definierar, och ge ett bibliotek som måste samexistera med andra ett eget scoped registry.

**Läget closed tas för ett skydd.** Läget closed döljer roten för värdens egenskap `shadowRoot` och döljer interna noder i en händelses composed path så som den ses av lyssnare utanför det slutna shadow tree-et. CSS Shadow Parts säger rent

ut att shadow trees medvetet inte är en säkerhetsgräns, bara en bekvämlighet för sidförfattare. Håll hemligheter utanför komponenterna, och välj läget closed för dess effekt på kod utifrån, inte som ett lås.

`attachShadow()` **anropas där det inte kan fungera.** DOM-standarden tillåter en shadow root bara på element med ett giltigt namn för custom elements och på en kort lista med inbyggda element, bland dem `div`, `section`, `p` och `span`, men inte `button`. På en customized built-in knapp kastar anropet ett `NotSupportedError`. Det beter sig också annorlunda på ett element som redan har en rot: är roten deklarativ och i samma läge tömmer anropet den och returnerar den, och annars kastar det ett fel. Kontrollera först om det redan finns en shadow root, så att markup som servern har skickat i en deklarativ rot behålls i stället för att raderas.

**Sidans stilar förväntas nå in.** CSS Shadow Parts kallar gränsen en stylingbarriär och påpekar att den också gör en komponent svårare att styla när sidan faktiskt vill det. Exponera de element som sidan får styla med `part`, och styla dem utifrån med `::part()`. En komponent inuti en annan behöver `exportparts` på den inre värden för att skicka sina delar vidare till sidan. Använd custom properties som sätts på värden för färger och avstånd som ska följa sidan; de ärvs in i komponentens shadow tree.

**Innehållet finns först efter skriptet.** En komponent som skriver all sin text från JavaScript visar inget av den för läsarna förrän definitionen har laddats, och ingenting alls om skriptet fallerar. Håll innehållet i light tree och tilldela det till slots, som produktkortet gör, så att uppgraderingen förbättrar det som redan finns där. Där innehåll måste vänta döljer HTML-standardens eget exempel det tills `customElements.whenDefined()` har uppfyllts för varje element som matchar `:not(:defined)`.

## Läs mer

- HTML Standard: Custom elements — namnregler, krav på konstruktorer, uppgraderingar, formulärkopplade element och `ElementInternals`, med genomgångna exempel.
- DOM Standard: Shadow tree — shadow roots och deras värdar, slots och de noder som tilldelas dem, och stegen bakom `attachShadow()`.
- HTML Standard: Scripting — elementet `template` med attributet `shadowrootmode`, och elementet `slot`.

- CSS Shadow Parts — W3C:s Working Draft från den 16 december 2025, som definierar `part`, `exportparts` och pseudoelementet `::part()`.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

**Se vad det kostar** Hör av er →

## Gör begreppen praktiska

Fråga hur ”Vad är web components?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

**Se våra tjänster**

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

**Omsorgsfullt skapad med kärlek av D.**

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

**Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.**

**Utskrivet från https://engineer.company/sv/glossary/web-components/**