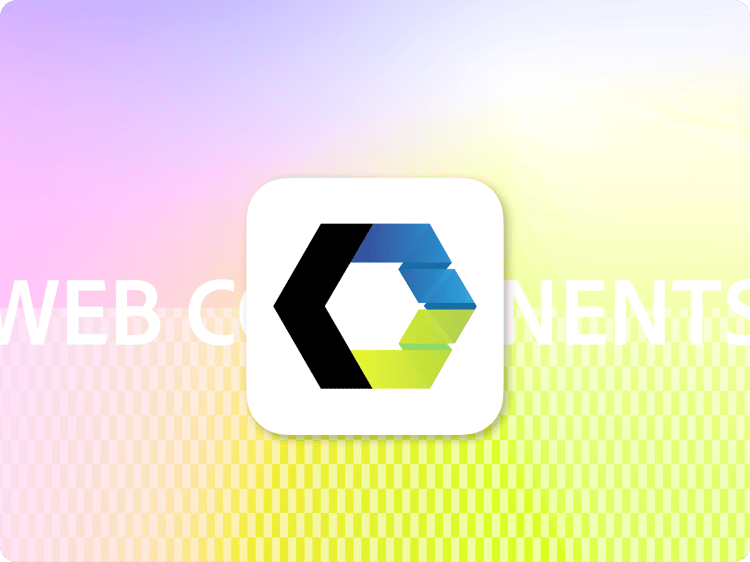
| Section Metadata | |
| --- | --- |
| style | content |



# Web Components

Web Components are a collection of web standards that allow the creation and use of reusable, modular functionality in web sites and web apps. They can be used in Adobe Experience Manager projects.

## Concepts and usage

There are three distinct web standards that make up [Web Components](https://developer.mozilla.org/en-US/docs/Web/API/Web_components). In order of relevance for AEM projects, these are:

1. [Custom elements](https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_custom_elements) allow the use of custom behavior and functionality in your existing site
2. [HTML templates](https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_templates_and_slots) enable the creation of DOM elements that are not displayed in the page
3. [Shadow DOM](https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_shadow_DOM) isolates a branch of the DOM tree from its parent, enabling re-use across a wide range of sites

Depending on how Web Components are written, their code and styles can be largely isolated from the rest of the page, which allows custom elements written by different teams to coexist on the same page, as well as sharing components between independent projects or teams.

They are often used for design systems in large organizations, but can also provide useful functionality such as dynamic loading of content, data retrieval from third-party sources, client-side content processing, etc.

## How to use Web Components in AEM projects?

To activate a Web Component, you need to load [the JavaScript code that defines it](https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_custom_elements) and have your AEM blocks code generate the corresponding custom HTML elements.

The code loading is similar to adding any existing JavaScript/CSS code to your project, but please read the “performance concerns” section below to avoid surprises.

### Example: page publication time

The following paragraph uses a *publication-time* AEM block that leverages GitHub’s [relative-time](https://github.com/github/relative-time-element) Web Component to display the page publication time in a friendly format:

| Publication Time (highlight) |
| --- |
| This page was published |

This would typically be used in the website footer if you want to indicate when each page was last published.

Reusing existing code makes total sense to compute friendly relative time strings like “two days ago”, “five hours ago”, so a Web Component is helpful, as our *publication-time* AEM block only needs to output something like:

<relative-time  
 datetime="2024-03-28T17:00:28.000Z">  
 28.03.2024  
</relative-time>

To let the *relative-time* Web Component do all the (somewhat) hard work.

From a coding standpoint, we only need to make the *relative-time* component code available and write the pretty simple glue code in the AEM *publication-time* block.

## Performance concerns, loading and lifecycle

To keep your Web Performance at the required level, your pages need to be frugal about how much JavaScript and CSS code they load, and if needed control when that loading occurs.

If your Web Components are written with performance in mind, there’s no reason for them to be less performant than AEM blocks, which are also backed by client-side JavaScript. Unfortunately, this cannot be taken for granted and you should test the performance of any Web Component before using it.

The AEM blocks code needs to translate the AEM semantic HTML to generate the custom HTML elements, but that’s no different from generating other HTML elements, so shouldn’t significantly affect performance.

If you want to use components from an existing library, you should make sure to load only JavaScript and CSS code that the current page actually requires, to avoid any extra baggage. Some libraries do that naturally, and others will require jumping through hoops to repackage their components in an optimized way. Your mileage may vary, and as always you’ll need to measure results to verify that the components code does not hamper performance.

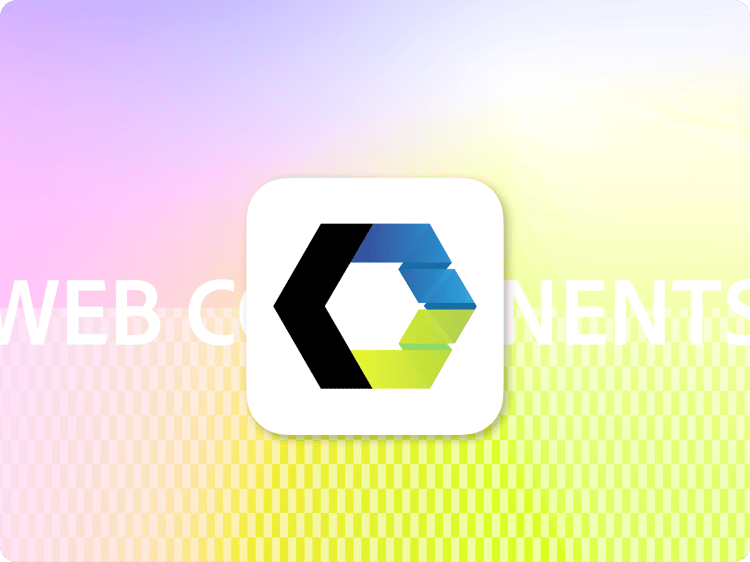
You also need to be aware of the AEM [Three-Phase-Loading principle](https://www.aem.live/developer/keeping-it-100#three-phase-loading-e-l-d), and if needed explicitly define when your Web Components code and styles are loaded.

One nice feature of Web Components that helps with lazy loading is that they take care of the asynchronicity and lifecycle concerns. Provided your components use the [lifecycle callbacks](https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_custom_elements) in the correct way, it does not matter whether their code becomes active before or after the corresponding elements are added to the DOM, the browser does the right thing.

| Metadata | |
| --- | --- |
| name | publication-time |

---

| Section Metadata | |
| --- | --- |
| style | content |



# Web Components

Web Components are a collection of web standards that allow the creation and use of reusable, modular functionality in web sites and web apps. They can be used in Adobe Experience Manager projects.

## Concepts and usage

There are three distinct web standards that make up [Web Components](https://developer.mozilla.org/en-US/docs/Web/API/Web_components). In order of relevance for AEM projects, these are:

1. [Custom elements](https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_custom_elements) allow the use of custom behavior and functionality in your existing site
2. [HTML templates](https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_templates_and_slots) enable the creation of DOM elements that are not displayed in the page
3. [Shadow DOM](https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_shadow_DOM) isolates a branch of the DOM tree from its parent, enabling re-use across a wide range of sites

Depending on how Web Components are written, their code and styles can be largely isolated from the rest of the page, which allows custom elements written by different teams to coexist on the same page, as well as sharing components between independent projects or teams.

They are often used for design systems in large organizations, but can also provide useful functionality such as dynamic loading of content, data retrieval from third-party sources, client-side content processing, etc.

## How to use Web Components in AEM projects?

To activate a Web Component, you need to load [the JavaScript code that defines it](https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_custom_elements) and have your AEM blocks code generate the corresponding custom HTML elements.

The code loading is similar to adding any existing JavaScript/CSS code to your project, but please read the “performance concerns” section below to avoid surprises.

### Example: page publication time

The following paragraph uses a *publication-time* AEM block that leverages GitHub’s [relative-time](https://github.com/github/relative-time-element) Web Component to display the page publication time in a friendly format:

| Publication Time (highlight) |
| --- |
| This page was published |

This would typically be used in the website footer if you want to indicate when each page was last published.

Reusing existing code makes total sense to compute friendly relative time strings like “two days ago”, “five hours ago”, so a Web Component is helpful, as our *publication-time* AEM block only needs to output something like:

<relative-time  
 datetime="2024-03-28T17:00:28.000Z">  
 28.03.2024  
</relative-time>

To let the *relative-time* Web Component do all the (somewhat) hard work.

From a coding standpoint, we only need to make the *relative-time* component code available and write the pretty simple glue code in the AEM *publication-time* block.

## Performance concerns, loading and lifecycle

To keep your Web Performance at the required level, your pages need to be frugal about how much JavaScript and CSS code they load, and if needed control when that loading occurs.

If your Web Components are written with performance in mind, there’s no reason for them to be less performant than AEM blocks, which are also backed by client-side JavaScript. Unfortunately, this cannot be taken for granted and you should test the performance of any Web Component before using it.

The AEM blocks code needs to translate the AEM semantic HTML to generate the custom HTML elements, but that’s no different from generating other HTML elements, so shouldn’t significantly affect performance.

If you want to use components from an existing library, you should make sure to load only JavaScript and CSS code that the current page actually requires, to avoid any extra baggage. Some libraries do that naturally, and others will require jumping through hoops to repackage their components in an optimized way. Your mileage may vary, and as always you’ll need to measure results to verify that the components code does not hamper performance.

You also need to be aware of the AEM [Three-Phase-Loading principle](https://www.aem.live/developer/keeping-it-100#three-phase-loading-e-l-d), and if needed explicitly define when your Web Components code and styles are loaded.

One nice feature of Web Components that helps with lazy loading is that they take care of the asynchronicity and lifecycle concerns. Provided your components use the [lifecycle callbacks](https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_custom_elements) in the correct way, it does not matter whether their code becomes active before or after the corresponding elements are added to the DOM, the browser does the right thing.

| Metadata | |
| --- | --- |
| name | publication-time (highlight) |