# WEB programming

## WEB Components

A Web Component is a way to create an encapsulated, single-responsibility code block that can be reused on any page. It plays the same role as a function in other programming languages. You can make a nice page header, name it **page-header** and use it on all pages just by writing a single element

<page-header></page-header>

Here is an example of the hello-world component (file **HelloWorld.js**):

class HelloWorld extends HTMLElement {

connectedCallback() {

this.textContent = 'Hello World';

}

}

customElements.define('hello-world', HelloWorld);

Each custom element is an ES6 class that must be binded to the **CustomElementRegistry**:

customElements.define('hello-world', HelloWorld);

The first parameter of this operator specifies a name of the element. By convention, it must be two words joined by a minus sign. The second parameter is a name of the class. Again, by convention, one must use the same words as were used in a name of custom element.

Here is an example of the **Autonomous custom elements**, the class inherits from **HTMLElement** in this case. Loading an element into a page is exactly the same as any other JS file:

<!DOCTYPE html>

<html>

<head>

<title>Page Title</title>

**<script type="module" src="./helloworld.js"></script>**

</head>

<body>

<h1>This is an example of the custom element</h1>

**<hello-world></hello-world>**

</body>

</html>

**Type="module"** may be omitted in the **script** element, but then add the **deferred** attribute:

<script src="./helloworld.js" defer></script>

A single element can also be used:

<hello-world/>

but form of two elements allows you to pass additional information into the component:

<hello-world>This text will be passed to the component</hello-world>

WEB component allows you to create **Customized built-in elements**. The base class can be any other element in that case.

class WordCount extends HTMLParagraphElement {

constructor() {

// Always call super first in constructor

super();

// Element functionality written in here

}

}

customElements.define('word-count', WordCount, { extends: 'p' });

In this case it is necessary to specify the base element when registering the component (see the object in the third parameter). In an HTML page, you specify the **base element** with the **is** attribute:

<p is="word-count"></p>

The custom element may be created using JavaScript. Use

document.createElement("hello-world")

**Autonomous custom element** requires a bit different operator:

document.createElement("p", { is: "word-count" })

Note that the first parameter specifies the **base** element and object in the second parameter defines the actual element.

#### Lifecycle of the component

Examples in the previous paragraph show two very important methods of custom element: **constructor()** and **connectedCallback()**. These two methods are called at different time:

* **constructor()** is called when the element is created; you can create the Shadow DOM here, but you can't add Nodes inside the normal DOM, and you can't add or set an attribute either.
* **connectedCallback()** is called when (after) the element is attached to the DOM.

Definion

A custom element is defined when customElements.define is called:

class HelloWorld extends HTMLElement {

connectedCallback() {

this.textContent = 'Hello World';

}

}

customElements.define('hello-world', HelloWorld);

Defining an element doesn't trigger either the constructor or the connectedCallback methods since it does not create an instance of an element. **An element can only be defined once.**

Create

An element can be created in JavaScript in two ways:

// can happen before definition

const myElement = document.createElement('my-element');

// can only happen if already defined

const myElement = new MyElement();

Creation triggers the **constructor**, if the element has already been defined. The constructor is called once per element instance.

Insert

An element is inserted into the DOM imperatively with JS:

document.body.append(myElement);

Any HTML element may be used instead of **document.body.** Insertion triggers the connectedCallback method, if the element has already been defined. Here is a trap:

<html>

<head>

<title>Page Title</title>

<!-- <script type="module" src="./helloworld.js"></script> -->

<script src="./helloworld.js"></script>

</head>

<body>

<h1>This is an example of the custom element</h1>

<p id="content">

</p>

<!-- <hello-world></hello-world> -->

<script>

const myElement = new HelloWorld();

parent = document.getElementById("content");

parent.append(myElement);

</script>

</body>

</html>

This HTML page will show nothing if you use

<script type="module" src="./helloworld.js"></script>

The browser defers module parsing, thus the **hello-world** element is undefined during the second script execution.

**connectedCallback** is called when the element is inserted into the DOM.

**disconnectedCallback** method called each time it is removed from the DOM

Declare

An element is declared when parsed as HTML. It may be done with HTML or JavaScript:

<my-element></my-element>

document.body.innerHTML = '<my-element></my-element>';

Declaration triggers the constructor and connectedCallback methods, if the element has already been defined.

Upgrade

The customElements **upgrade**() method upgrades all shadow-containing custom elements of the document in a node subtree, even before they are connected to the main document.

customElements.upgrade(root);

This method returns void. Parameter **root** is a node instance with shadow-containing descendant elements that are to be upgraded.

#### CustomElementRegistry

[A Complete Introduction to Web Components in 2023 (kinsta.com)](https://kinsta.com/blog/web-components/)

[Using custom elements - Web APIs | MDN (mozilla.org)](https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_custom_elements)

[javascript - Difference between constructor and connectedCallback in custom elements v1 - Stack Overflow](https://stackoverflow.com/questions/40492330/difference-between-constructor-and-connectedcallback-in-custom-elements-v1/40494899" \l "40494899)