In this article

Sanitizer

Limited availability

This feature is not Baseline because it does not work in some of the most widely-used browsers.

Want more browser support for this feature? Tell us why.

Learn more | See full compatibility

The **`Sanitizer`** interface of the HTML Sanitizer API defines a configuration object that specifies what elements, attributes and comments are allowed or should be removed when inserting strings of HTML into an `Element` or `ShadowRoot`, or when parsing an HTML string into a `Document`.

Constructors

`Sanitizer()`

Creates and returns a `Sanitizer` object, optionally with custom sanitization behavior defined in a `SanitizerConfig`.

Instance methods

`Sanitizer.allowElement()`

Sets an element as allowed by the sanitizer, optionally with an array of attributes that are allowed or disallowed.

`Sanitizer.get()`

Returns the current `Sanitizer` configuration as a `SanitizerConfig` dictionary instance.

`Sanitizer.removeElement()`

Sets an element to be removed by the sanitizer.

`Sanitizer.removeUnsafe()`

Updates the sanitizer configuration so that it will remove any XSS-unsafe HTML.

`Sanitizer.replaceElementWithChildren()`

Sets an element to be replaced by its child HTML elements.

`Sanitizer.allowAttribute()`

Sets an attribute as allowed on any element.

`Sanitizer.removeAttribute()`

Sets an attribute to be removed from any element.

`Sanitizer.setComments()`

Sets whether comments will be allowed or removed by the sanitizer.

`Sanitizer.setDataAttributes()`

Sets whether data attributes on elements will be allowed or removed by the sanitizer.

Description

A `Sanitizer` is a reusable configuration object that specifies what elements, attributes and comments are allowed or should be removed when inserting strings of HTML into an `Element` or `ShadowRoot`, or when parsing an HTML string into a `Document`. It can be used with the following sanitization methods:

- Safe methods: `Element.setHTML()`, `ShadowRoot.setHTML()`, and `Document.parseHTML()`.
- Unsafe methods: `Element.setHTMLUnsafe()`, `ShadowRoot.setHTMLUnsafe()`, and `Document.parseHTMLUnsafe()`.

A `Sanitizer` instance can be constructed from a `SanitizerConfig`, and is effectively a wrapper around that object. A `Sanitizer` and a `SanitizerConfig` can be used with the same methods, but if you're using the same configuration multiple times, it's expected to be more efficient to use a `Sanitizer` and modify it when you need to. If no `SanitizerConfig` is passed to the constructor, the sanitizer is created with the default sanitizer configuration, which removes XSS-unsafe elements and attributes, along with other elements and attributes that can potentially be used in other attacks, such as clickjacking and spoofing.

Note that any `Sanitizer` can be made XSS-safe by calling `Sanitizer.removeUnsafe()`, but other potentially dangerous elements and attributes — which are removed by the default configuration — may still be present.

Using `Sanitizer` with the sanitization methods

The default `Sanitizer` configuration is automatically used if no `Sanitizer` is passed to `Element.setHTML` or the other safe sanitization methods. This is a reasonable default as it restricts the attack surface while still allowing the majority of use cases.

If a custom sanitizer is passed to these methods, any XSS-unsafe elements and attributes allowed by the sanitizer would also be removed. Note that although this removes the same elements as `Sanitizer.removeUnsafe()`, the passed `Sanitizer` is not actually changed by the operation: it would be "unsafe" if used in an unsafe method.

The unsafe sanitization methods perform no sanitization by default. However, as noted, you can call `Sanitizer.removeUnsafe()` to remove any XSS-unsafe elements (if you want to use the default configuration, you should use the safe methods).

Examples

For more examples see the HTML Sanitizer API and the individual methods. Below we show a few examples of how you might create different sanitizer configurations.

Creating a default sanitizer

The default sanitizer is constructed as shown below.

JS Copy

```
const sanitizer = new Sanitizer();
```

The XSS-safe sanitization methods create the same sanitizer automatically if no options are passed.

Creating an empty sanitizer

To create an empty sanitizer, pass an empty object to the constructor. The resulting sanitizer configuration is a remove configuration with empty arrays.

JS Copy

```
const sanitizer = new Sanitizer({});
console.log(sanitizer.get());
/*
{
  "comments": true,
  "removeAttributes": [],
  "removeElements": []
}
*/
```

Creating an "allow" sanitizer

This example shows how you might create an "allow sanitizer": a sanitizer that allows only a subset of attributes and elements.

The code first uses the `Sanitizer()` constructor to create a `Sanitizer`, specifying a `SanitizerConfig` that allows the element `<div>`, `<p>` and `<script>`.

The example then uses `allowElement()` to further allow `<span>` elements, `allowAttribute()` to allow the `id` attribute on any element, and `replaceElementWithChildren()` method to set that any `<b>` elements should be replaced by their inner content (this is a kind of "allow" in that you are explicitly specifying some entities to keep). Lastly we specify that comments should be retained.

JS Copy

```
const sanitizer = new Sanitizer({ elements: ["div",
"p", "script"] });
sanitizer.allowElement("span");
sanitizer.allowAttribute("id");
sanitizer.replaceElementWithChildren("b");
sanitizer.setComments(true);
```

Creating a "remove" sanitizer

This example shows how you might create a "remove sanitizer", specifying items to remove from the input.

The code first uses the `Sanitizer()` constructor to create a `Sanitizer`, specifying a `SanitizerConfig` that removes the element `<span>` and `<script>`. We then use `removeElement()` to add `<h6>` to the array of elements to be removed, and `removeAttribute()` to remove `lang` from the attributes list. We also remove comments.

JS Copy

```
const sanitizer = new Sanitizer({ removeElements:
["span", "script"] });
sanitizer.removeElement("h6");
sanitizer.removeAttribute("lang");
sanitizer.setComments(false);
```

Specifications

Specification
HTML # sanitizer

Browser compatibility

Report problems with this compatibility data • View data on GitHub

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Browser	WebView Android	WebView on iOS
`Sanitizer`	146	146	148	130	No	146	148	97	No	No	146	No
`Sanitizer()` constructor	146	146	148	130	No	146	148	97	No	No	146	No
`allowAttribute`	146	146	148	130	No	146	148	97	No	No	146	No
`allowElement`	146	146	148	130	No	146	148	97	No	No	146	No
`allowProcessingInstruction`	150	150	No	134	No	150	No	No	No	No	150	No
`get`	146	146	148	130	No	146	148	97	No	No	146	No
`removeAttribute`	146	146	148	130	No	146	148	97	No	No	146	No
`removeElement`	146	146	148	130	No	146	148	97	No	No	146	No
`removeProcessingInstruction`	150	150	No	134	No	150	No	No	No	No	150	No
`removeUnsafe`	146	146	148	130	No	146	148	97	No	No	146	No
`replaceElementWithChildren`	146	146	148	130	No	146	148	97	No	No	146	No
`setComments`	146	146	148	130	No	146	148	97	No	No	146	No
`setDataAttributes`	146	146	148	130	No	146	148	97	No	No	146	No

Tip: you can click/tap on a cell for more information.

Full support

No support

Experimental. Expect behavior to change in the future.

See implementation notes.