Document: querySelector() method

- Syntax
  - Parameters
  - Return value
  - Exceptions
- Examples
  - Finding the first element matching a class
  - Complex selectors
  - Negation
  - Escaping attribute values
    - HTML
    - CSS
    - JavaScript
    - Result
- Specifications
- Browser compatibility
- See also

{{ApiRef("DOM")}}

The {{domxref("Document")}} method `querySelector()` returns the first {{domxref("Element")}} within the document that matches the specified CSS selector, or group of CSS selectors. If no matches are found, `null` is returned.

The matching is done using depth-first pre-order traversal of the document's nodes starting with the first element in the document's markup and iterating through sequential nodes by order of the number of child nodes.

If the specified selector matches an ID that is incorrectly used more than once in the document, the first element with that ID is returned.

CSS pseudo-elements will never return any elements.

Syntax

```
querySelector(selectors)
```

Parameters

- `selectors`
  - : A string containing one or more selectors to match. This string must be a valid CSS selector string; if it isn't, a `SyntaxError` exception is thrown.
    Note that the HTML specification does not require attribute values to be valid CSS identifiers. If a `class` or `id` attribute value is not a valid CSS identifier, then you must escape it before using it in a selector, either by calling {{domxref("CSS.escape_static", "CSS.escape()")}} on the value, or using one of the techniques described in Escaping characters. See Escaping attribute values for an example.

Return value

An {{domxref("Element")}} object representing the first element in the document that matches the specified set of CSS selectors, or `null` if there are no matches.

If you need a list of all elements matching the specified selectors, you should use {{domxref("Document.querySelectorAll", "querySelectorAll()")}} instead.

Exceptions

- `SyntaxError` {{domxref("DOMException")}}
  - : Thrown if the syntax of the specified *selectors* is invalid.

Examples

Finding the first element matching a class

In this example, the first element in the document with the class `myclass` is returned:

```
const el = document.querySelector(".myclass");
```

Complex selectors

Selectors can also be really powerful, as demonstrated in the following example. Here, the first {{HTMLElement("input")}} element with the name "login" (`<input name="login"/>`) located inside a {{HTMLElement("div")}} whose class is "user-panel main" (`<div class="user-panel main">`) in the document is returned:

```
const el = document.querySelector("div.user-panel.main input[name='login']");
```

Negation

As all CSS selector strings are valid, you can also negate selectors:

```
const el = document.querySelector(
  "div.user-panel:not(.main) input[name='login']",
);
```

This will select an input with a parent div with the `user-panel` class but not the `main` class.

Escaping attribute values

This example shows that if an HTML document contains an `id` which is not a valid CSS identifier, then we must escape the attribute value before using it in `querySelector()`.

HTML

In the following code, a {{htmlelement("div")}} element has an `id` of `"this?element"`, which is not a valid CSS identifier, because the `"?"` character is not allowed in CSS identifiers.

We also have three buttons, and a {{htmlelement("pre")}} element for logging errors.

```
<div id="this?element"></div>
```

```
<button id="no-escape">No escape</button>
<button id="css-escape">CSS.escape()</button>
<button id="manual-escape">Manual escape</button>

<pre id="log"></pre>
```

CSS

```
div {
  background-color: blue;
  margin: 1rem 0;
  height: 100px;
  width: 200px;
}
```

JavaScript

All three buttons, when clicked, try to select the `<div>`, and then set its background color to a random value.

- The first button uses the `"this?element"` value directly.
- The second button escapes the value using {{domxref("CSS.escape_static", "CSS.escape()")}}.
- The third button explicitly escapes the `"?"` character using a backslash. Note that we must also escape the backslash itself, using another backslash, like: `"\\?"`.

```
const log = document.querySelector("#log");

function random(number) {
  return Math.floor(Math.random() * number);
}

function setBackgroundColor(id) {
  log.textContent = "";

  try {
    const element = document.querySelector(`#${id}`);
    const randomColor = `rgb(${random(255)} ${random(255)} ${random(255)})`;
    element.style.backgroundColor = randomColor;
  } catch (e) {
    log.textContent = e;
```

```
  }
}

document.querySelector("#no-escape").addEventListener("click", () => {
  setBackgroundColor("this?element");
});

document.querySelector("#css-escape").addEventListener("click", () => {
  setBackgroundColor(CSS.escape("this?element"));
});

document.querySelector("#manual-escape").addEventListener("click", () => {
  setBackgroundColor("this\\?element");
});
```

Result

Clicking the first button gives an error, while the second and third buttons work properly.

{{embedlivesample("escaping_attribute_values", "", 200)}}

Specifications

{{Specifications}}

Browser compatibility

{{Compat}}

See also

- Selection and traversal on the DOM tree
- {{domxref("Element.querySelector()")}}
- {{domxref("Document.querySelectorAll()")}}
- {{domxref("Element.querySelectorAll()")}}

□□□ #2
□ □□□ □□□ 2026-10-31 03:27:53 UTC

□ □□□ □□□ 2026-10-31 03:27:53 UTC