Element: computedStyleMap() method

- Syntax
  - Parameters
  - Return value
- Examples
  - Getting default styles
  - Comparison with getComputedStyle()
- Specifications
- Browser compatibility
- See also

The `computedStyleMap()` method of the Element interface returns a StylePropertyMapReadOnly interface which provides a read-only representation of a CSS declaration block that is an alternative to CSSStyleDeclaration.

Syntax

```
computedStyleMap()
```

Parameters

None.

Return value

A StylePropertyMapReadOnly object.

Unlike Window.getComputedStyle, the return value contains computed values, not resolved values. For most properties, they are the same, except a few layout-related properties, where the resolved value is the used value instead of the computed value. See the comparison with `getComputedStyle()` example for details.

Examples

Getting default styles

We start with some simple HTML: a paragraph with a link, and a definition list to which we will add all the CSS Property / Value pairs.

```
<p>
  <a href="https://example.com">Link</a>
</p>
<dl id="regurgitation"></dl>
```

We add a little bit of CSS

```
a {
  --color: red;
  color: var(--color);
}
```

We add JavaScript to grab our link and return a definition list of all the CSS property values using `computedStyleMap()`.

```
// get the element
const myElement = document.querySelector("a");

// get the <dl> we'll be populating
const stylesList = document.querySelector("#regurgitation");

// Retrieve all computed styles with computedStyleMap()
const allComputedStyles = myElement.computedStyleMap();

// iterate through the map of all the properties and values, adding a <dt> and <dd> for each
for (const [prop, val] of allComputedStyles) {
```

```
  // properties
  const cssProperty = document.createElement("dt");
  cssProperty.appendChild(document.createTextNode(prop));
  stylesList.appendChild(cssProperty);

  // values
  const cssValue = document.createElement("dd");
  cssValue.appendChild(document.createTextNode(val));
  stylesList.appendChild(cssValue);
}
```

In browsers that support `computedStyleMap()`, you'll see a list of all the CSS properties and values. In other browsers you'll just see a link.

Did you realize how many default CSS properties a link had? Update the `document.querySelector("a")` to `document.querySelector("p")`, and you'll notice a difference in the `margin-top` and `margin-bottom` default computed values.

Comparison with getComputedStyle()

Window.getComputedStyle() returns resolved values, while `computedStyleMap()` returns computed values. These are usually the same, but for some properties, the resolved value is the used value instead of the computed value. For example, percentage values for widths are resolved to pixel values *post-layout*, so the used values are in pixels, while the computed values are still in percentages.

Note that the way we present it makes the two APIs seem more similar than they are. `computedStyleMap()` contains CSS Typed OM objects, while `getComputedStyle()` contains strings. The former presents the same information in a more structured and processable way.

In this example, the `width` property is specified as a percentage, so the computed value is given as a percentage, but the resolved value is given in pixels. The `height` is always in pixels. The `background-color` is a named color, but it is computed to an RGB value.

```
<div class="container">
  <div class="item"></div>
</div>
<pre id="result"></pre>
```

```
.container {
  width: 200px;
```

```
  height: 200px;
}

.item {
  width: 50%;
  height: 100px;
  background-color: tomato;
}
```

```
const item = document.querySelector(".item");
const result = document.querySelector("#result");
const resolvedValues = getComputedStyle(item);
const computedValues = item.computedStyleMap();

result.textContent = `resolvedValues.width = ${resolvedValues.width}
computedValues.get("width") = ${computedValues.get("width")}

resolvedValues.height = ${resolvedValues.height}
computedValues.get("height") = ${computedValues.get("height")}

resolvedValues.backgroundColor = ${resolvedValues.backgroundColor}
computedValues.get("background-color") = ${computedValues.get(
  "background-color",
)}`;
```

Specifications

Browser compatibility

See also

- Window.getComputedStyle()

□□□ #3
□ □□□ □□□ 2026-10-31 03:27:53 UTC
□ cocobook-sync □□□ 2026-11-04 17:34:19 UTC