

Constructing the Object Model



- [Summary {:.hide-from-toc }](#)
 - [Document Object Model \(DOM\)](#)
 - [CSS Object Model \(CSSOM\)](#)
 - [Feedback {:#feedback }](#)
-

Before the browser can render the page, it needs to construct the DOM and CSSOM trees. As a result, we need to ensure that we deliver both the HTML and CSS to the browser as quickly as possible.

Summary {:.hide-from-toc }

- Bytes → characters → tokens → nodes → object model.
- HTML markup is transformed into a Document Object Model (DOM); CSS markup is transformed into a CSS Object Model (CSSOM).
- DOM and CSSOM are independent data structures.
- Chrome DevTools Performance panel allows us to capture and inspect the construction and processing costs of DOM and CSSOM.

Document Object Model (DOM)

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width,initial-scale=1" />
    <link href="style.css" rel="stylesheet" />
    <title>Critical Path</title>
  </head>
  <body>
```

```
<p>Hello <span>web performance</span> students!</p>
<div></div>
</body>
</html>
```

[Try it](#)

Let's start with the simplest possible case: a plain HTML page with some text and a single image. How does the browser process this page?

```
{% Img src="image/C47gYyWYVMMhDmtYSLOWazuyePF2/cSL20piziX7XLekCPCuD.png", alt="DOM construction process", width="800", height="443" %}
```

1. **Conversion:** The browser reads the raw bytes of HTML off the disk or network, and translates them to individual characters based on specified encoding of the file (for example, UTF-8).
2. **Tokenizing:** The browser converts strings of characters into distinct tokens—as specified by the [W3C HTML5 standard](#) for example, "<html>", "<body>"—and other strings within angle brackets. Each token has a special meaning and its own set of rules.
3. **Lexing:** The emitted tokens are converted into "objects," which define their properties and rules.
4. **DOM construction:** Finally, because the HTML markup defines relationships between different tags (some tags are contained within other tags) the created objects are linked in a tree data structure that also captures the parent-child relationships defined in the original markup: the *HTML* object is a parent of the *body* object, the *body* is a parent of the *paragraph* object, and so on.

```
{% Img src="image/C47gYyWYVMMhDmtYSLOWazuyePF2/keK3wDv9k2KzJA9QubFx.png", alt="DOM tree", width="582", height="299" %}
```

The final output of this entire process is the Document Object Model (DOM) of our simple page, which the browser uses for all further processing of the page.

Every time the browser processes HTML markup, it goes through all of the steps above: convert bytes to characters, identify tokens, convert tokens to nodes, and build the DOM tree. This entire process can take some time, especially if we have a large amount of HTML to process.

```
{% Img src="image/C47gYyWYVMMhDmtYSLOWazuyePF2/RHpJTxB4gBYuhzILaB7Y.png", alt="Tracing DOM construction in DevTools", width="766", height="180" %}
```

{% Aside %} We're assuming that you have basic familiarity with Chrome DevTools - that is, you know how to capture a network waterfall or record a timeline. If you need a quick refresher, check out the [Chrome DevTools documentation](#); if you're new to DevTools, check out [DevTools for Beginners](#) in our DevTools documentation. {% endAside %}

If you open up Chrome DevTools and record a timeline while the page is loaded, you can see the actual time taken to perform this step—in the example above, it took us ~5ms to convert a chunk of HTML into a DOM tree. For a larger page, this process could take significantly longer. When creating smooth animations, this can easily become a bottleneck if the browser has to process large amounts of HTML.

The DOM tree captures the properties and relationships of the document markup, but it doesn't tell us how the element will look when rendered. That's the responsibility of the CSSOM.

CSS Object Model (CSSOM)

While the browser was constructing the DOM of our simple page, it encountered a link tag in the head section of the document referencing an external CSS stylesheet: `style.css`. Anticipating that it needs this resource to render the page, it immediately dispatches a request for this resource, which comes back with the following content:

```
body {  
  font-size: 16px;  
}  
p {  
  font-weight: bold;  
}  
span {  
  color: red;  
}  
p span {  
  display: none;  
}  
img {  
  float: right;  
}
```

We could have declared our styles directly within the HTML markup (inline), but keeping our CSS independent of HTML allows us to treat content and design as separate concerns: designers can work on CSS, developers can focus on HTML, and so on.

As with HTML, we need to convert the received CSS rules into something that the browser can understand and work with. Hence, we repeat the HTML process, but for CSS instead of HTML:

```
{% Img src="image/C47gYyWYVMMhDmtYSLOWazuyePF2/yb5YfU0vx6vHvB7c0fyD.png",  
alt="CSSOM construction steps", width="800", height="51" %}
```

The CSS bytes are converted into characters, then tokens, then nodes, and finally they are linked into a tree structure known as the "CSS Object Model" (CSSOM):

```
{% Img src="image/C47gYyWYVMMhDmtYSLOWazuyePF2/keK3wDv9k2KzJA9QubFx.png", alt="CSSOM tree", width="582", height="299" %}
```

Why does the CSSOM have a tree structure? When computing the final set of styles for any object on the page, the browser starts with the most general rule applicable to that node (for example, if it is a child of a body element, then all body styles apply) and then recursively refines the computed styles by applying more specific rules; that is, the rules "cascade down."

To make it more concrete, consider the CSSOM tree above. Any text contained within the `<span>` tag that is placed within the body element, has a font size of 16 pixels and has red text—the `font-size` directive cascades down from the `body` to the `span`. However, if a `span` tag is child of a paragraph (`p`) tag, then its contents are not displayed.

Also, note that the above tree is not the complete CSSOM tree and only shows the styles we decided to override in our stylesheet. Every browser provides a default set of styles also known as "user agent styles"—that's what we see when we don't provide any of our own—and our styles simply override these defaults.

To find out how long the CSS processing takes you can record a timeline in DevTools and look for "Recalculate Style" event: unlike DOM parsing, the timeline doesn't show a separate "Parse CSS" entry, and instead captures parsing and CSSOM tree construction, plus the recursive calculation of computed styles under this one event.

```
{% Img src="image/C47gYyWYVMMhDmtYSLOWazuyePF2/nAl6iO8vsAf03EUs4nWZ.png", alt="Tracing CSSOM construction in DevTools", width="766", height="180" %}
```

Our trivial stylesheet takes ~0.6ms to process and affects eight elements on the page—not much, but once again, not free. However, where did the eight elements come from? The CSSOM and DOM are independent data structures! Turns out, the browser is hiding an important step. Next, let's talk about the [render tree](#) that links the DOM and CSSOM together.

Feedback { : #feedback }

👤 #2

🕒 2026-10-31 21:46:35 UTC

🕒 2026-10-31 21:46:37 UTC