**Web page rendering**

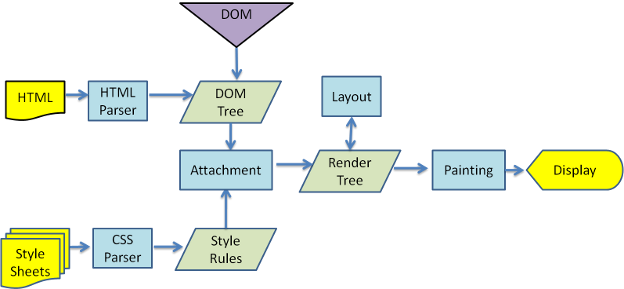
Rendering is a process used in web development that turns website code into the interactive pages users see when they visit a website. The term generally refers to the use of HTML, CSS, and JavaScript codes. The process is completed by a rendering engine, the software used by a web browser to render a web page.

## How browsers render web pages

Web browsers render web pages in the following sequence:

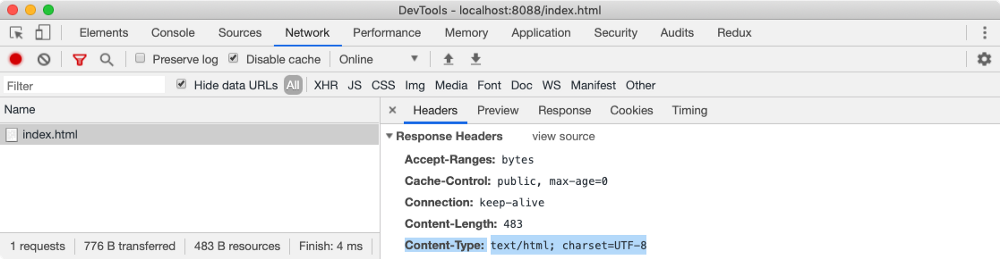
**Constructs DOM and CSSOM from raw code**

* While loading a web page, a web server sends a folder of files containing HTML, CSS, and JavaScript code to a user’s web browser.
* The browser engine converts this data (bytes) into characters (the HTML code).
* It parses the characters into tokens[[1]](https://www.seobility.net/en/wiki/Rendering#cite_note-1), which are further parsed into nodes[[2]](https://www.seobility.net/en/wiki/Rendering#cite_note-2).
* The browser engine links the nodes into a tree-like structure known as a Document Object Model (DOM). The DOM is the JavaScript representation of the HTML.
* Simultaneously, the browser converts the CSS code to a CSS Object Model (CSSOM) through a similar process.



When a browser sends a request to a server to fetch an HTML document, the server returns an HTML page in binary stream format which is basically a text file with the response header Content-Type set to the value text/html; charset=UTF-8.

Here text/html is a MIME Type which tells the browser that it is an HTML document and charset=UTF-8 tells the browser that it is encoded in **UTF-8** character encoding. Using this information, the browser can convert the binary format into a readable text file.



If this header is missing, the browser would not understand how to process the file and it will render in plain text format. But if everything is OK, after this conversion, the browser can start reading the HTML document.

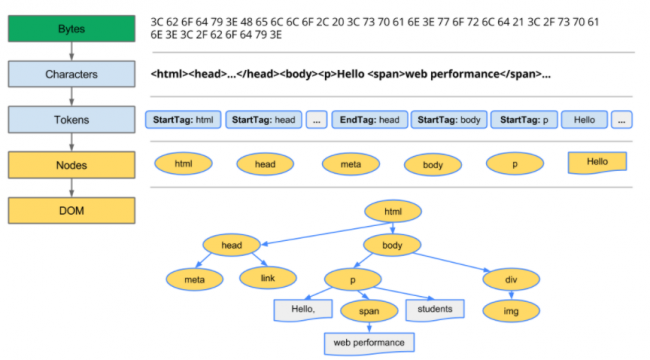
Our webpage is dependent on style.css to provide styles to HTML elements and main.js to perform some JavaScript operations.

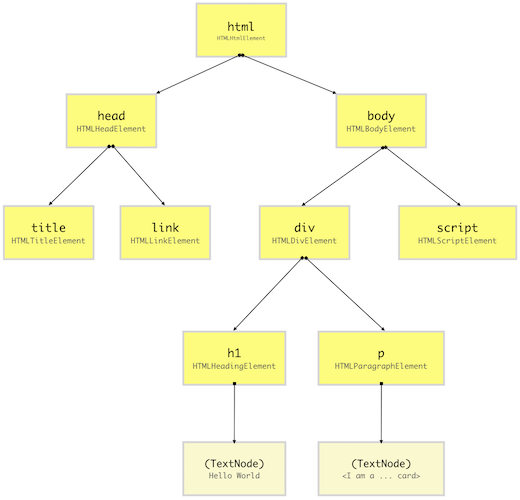
**Document Object Model (DOM)**

When the browser reads HTML code, whenever it encounters an HTML element like html, body, div etc., it creates a JavaScript object called a Node. Eventually, all HTML elements will be converted to JavaScript objects.

Since every HTML element has different properties, the Node object will be created from different classes (constructor functions). For example, the Node object for the div element is created from HTMLDivElement which inherits Node class. The browser comes with built-in classes like HTMLDivElement, HTMLScriptElement, Node etc.

After the browser has created Nodes from the HTML document, it has to create a tree-like structure of these node objects. Since our HTML elements in the HTML file are nested inside each other, the browser needs to replicate that but using Node objects it has previously created. This will help the browser efficiently render and manage the webpage throughout its lifecycle.



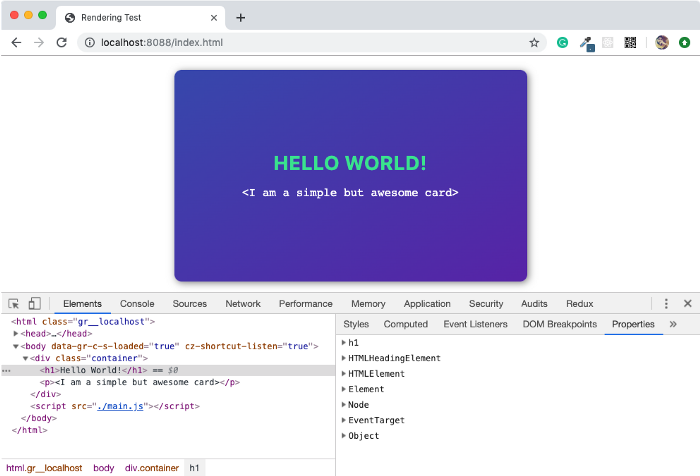


(**DOM Tree**)

A DOM tree starts from the topmost element which is html element and branches out as per the occurrence and nesting of HTML elements in the document. Whenever an HTML element is found, it creates a DOM node (Node) object from its respective class (constructor function).

A DOM node doesn’t always have to be an HTML element. When the browser creates a DOM tree, it also saves things like comments, attributes, text as separate nodes in the tree.

You can visualize the DOM tree in Google Chrome DevTools Console as shown below. This will show you the hierarchy of DOM elements (a high-level view of DOM tree) with properties of each DOM element.



JavaScript doesn’t understand what DOM is, it is not part of the JavaScript specifications. DOM is a high-level Web API provided by the browser to efficiently render a webpage and expose it publically for the developer to dynamically manipulate DOM elements for various purposes.

Using DOM API, developers can add or remove HTML elements, change its appearance or bind event listeners. Using DOM API, HTML elements can be created or cloned in memory and maniuplated without affecting the rendered DOM tree. This gives developers the ability to construct highly dynamic web page with rich user experience.

**CSS Object Model (CSSOM)**

When we design a website, our intentions are to make it as good looking as possible. And we do that by providing some styles to HTML elements. In the HTML page, we provide styles to HTML elements using CSS which stands for Cascading Style Sheets. Using CSS selectors, we can target DOM elements and set a value to style property such as color or font-size.

There are different methods of applying styles to HTML elements like using an external CSS file, with embedded CSS using <style> tag, with an inline method using the style attribute on HTML elements or using JavaScript. But in the end, the browser has to do the heavy lifting of applying CSS styles to the DOM elements.

After constructing the DOM, the browser reads CSS from all the sources (external, embedded, inline, user-agent, etc.) and construct a CSSOM. CSSOM stands for CSS Object Model which is a Tree Like structure just like DOM.

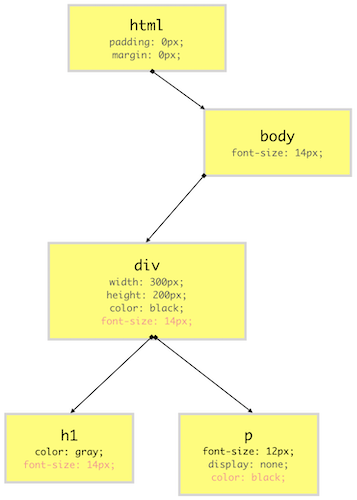
Each node in this tree contains CSS style information that will be applied to DOM elements that it target (specified by the selector). CSSOM, however, does not contain DOM elements which can’t be printed on the screen like <meta>, <script>, <title> etc.

As we know, most of the browser comes with its own stylesheet which is called as user agent stylesheet, the browser first computes final CSS properties for DOM element by overriding user agent styles with CSS provided by the developer properties (using specificity rules) and then construct a node.

Even if a CSS property (such as display) for a particular HTML element isn’t defined by either the developer or the browser, its value is set to the default value of that property as specified by the W3C CSS standard. While selecting the default value of a CSS property, some rules of inheritance are used if a property qualifies for the inheritance as mentioned in the W3C documentation.

For example, color and font-size among others inherits the value of the parent if these properties are missing for an HTML element. So you can imagine having these properties on an HTML element and all its children inheriting it. This is called cascading of styles and that’s why CSS is an acronym of Cascading Style Sheets. This is the very reason why the browser constructs a CSSOM, a tree-like structure to compute styles based on CSS cascading rules.

For example, color and font-size among others inherits the value of the parent if these properties are missing for an HTML element. So you can imagine having these properties on an HTML element and all its children inheriting it. This is called cascading of styles and that’s why CSS is an acronym of Cascading Style Sheets. This is the very reason why the browser constructs a CSSOM, a tree-like structure to compute styles based on CSS cascading rules.



(**CSSOM Tree**)

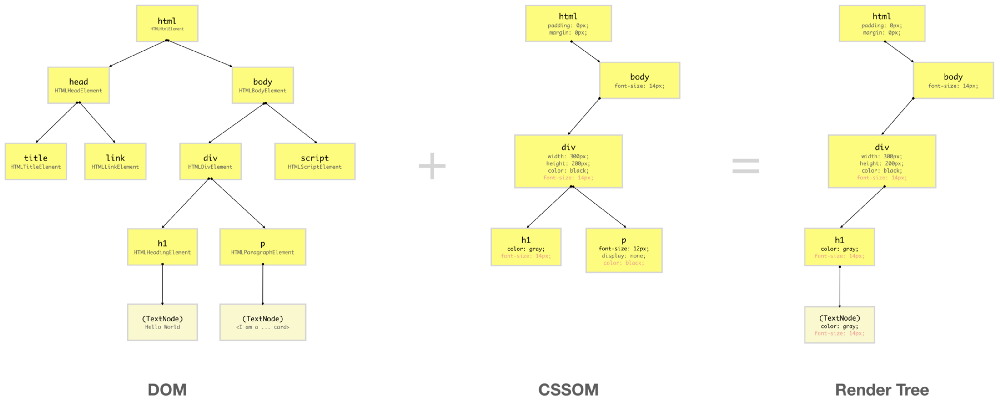
our CSSOM tree does not contain elements which do not get printed on the screen like <link>, <title>, <script> etc. CSS property values in the red colors are cascaded down from the top while property values in the gray are overriding the inherited values.

## ****Render Tree****

Render-Tree is also a tree-like structure constructed by combining DOM and CSSOM trees together. The browser has to calculate the layout of each visible element and paint them on the screen, for that browser uses this Render-Tree. Hence, unless Render-Tree isn’t constructed, nothing is going to get printed on the screen which is why we need both DOM and CSSOM trees.

As Render-Tree is a low-level representation of what will eventually get printed on the screen, it won’t contain nodes that do not hold any area in the pixel matrix. For example, display:none; elements have dimensions of 0px X 0px, hence they won’t be present in Render-Tree.

Unlike DOM API which gives access to the DOM elements in the DOM tree constructed by the browser, CSSOM is kept hidden from the user. But since the browser combines DOM and CSSOM to form the Render Tree, the browser exposes the CSSOM node of a DOM element by providing high-level API on the DOM element itself. This enables the developer to access or change the CSS properties of a CSSOM node.



# **Rendering Sequence**

When a web page is loaded, the browser first reads the HTML text and constructs DOM Tree from it. Then it processes the CSS whether that is inline, embedded, or external CSS and constructs the CSSOM Tree from it.

After these trees are constructed, then it constructs the Render-Tree from it. Once the Render-Tree is constructed, then the browser starts the printing individual elements on the screen.

## 1.Layout operation

The first browser creates the layout of each individual Render-Tree node. The layout consists of the **size of each node in pixels and where (position) it will be printed on the screen**. This process is called layout since the browser is calculating the layout information of each node.

This process is also called **reflow** or browser reflow and it can also occur when you **scroll, resize** the window or manipulate DOM elements. Here is a list of events that can trigger the layout/reflow of the elements.

## **2.Paint operation**

Until now we have a list of geometries that need to be printed on the screen. Since elements (or a sub-tree) in the Render-Tree can overlap each other and they can have CSS properties that make them frequently change the look, position, or geometry (such as animations), the browser creates a **layer** for it.

Creating layers helps the browser efficiently perform painting operations throughout the lifecycle of a web page such as while scrolling or resizing the browser window. Having layers also help the browser correctly draw elements in the stacking order (along the z-axis) as they were intended by the developer.

Now that we have layers, we can combine them and draw them on the screen. But the browser does not draw all the layers in a single go. Each layer is drawn separately first.

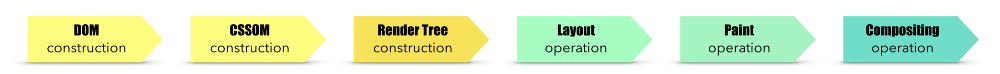
Inside each layer, the browser fills the individual pixels for whatever visible property the element has such as border, background color, shadow, text, etc. This process is also called as **rasterization**. To increase performance, the browser may use different threads to perform rasterization.

You can visualize different layers on a web page from Chrome DevTools. Open DevTools and from more tools options, select Layers. You can also visualize layer borders from the Rendering panel.

## 3.Compositing operation

Until now, we haven’t drawn a single pixel on the screen. What we have are different layers (bitmap images) that should be drawn on the screen in a specific order. In compositing operations, these layers are sent to GPU to finally draw it on the screen.

Sending entire layers to draw is clearly inefficient because this has to happen every time there is a reflow (layout) or repaint. Hence, a layer is broken down into different tiles which then will be drawn on the screen. You can also visualize these tiles in Chrome’s DevTool Rendering panel.



## 4.Browser engines

The job of creating DOM Tree, CSSOM Tree, and handle rendering logic is done using a piece of software called a Browser Engine (also known as Rendering Engine or Layout Engine) which resides inside the browser. This browser engine contains all the necessary elements and logic to render a web page from HTML code to actual pixels on the screen.

If you heard people talking about WebKit, they were talking about a browser engine. WebKit is used by Apple’s Safari browser and was the default rendering engine for the Google Chrome browser. As of now, the Chromium project uses Blink as the default rendering engine.