

|  |
| --- |
| Week 1  Assignment-1 |
|  |
| April 2  Web Browser Functionalities  Authored by: Shubham Patil |



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Main components of web browser:  * User Interface * Browser engine * Rendering engine * Networking * JavaScript interpreter * Data Storage  1. User Interface It is an environment allowing users to use certain features like search bar, refresh button, menu, bookmarks, etc. The user interface provides the methods with which a user interacts with the browser Engine. Every part of the browser display except the window where you see the requested page. 2. Browser Engine The bridge connects the interface and the engine. It monitors the rendition engine while manipulating the inputs coming from multiple user interfaces. The Browser Engine provides methods to initiate the loading of a URL and other high-level browsing actions (reload, back, forward). The Browser Engine also provides the User interface with various messages relating to error messages and loading progress. 3. The Rendering Engine The Rendering Engine produces the visual representation of a given URL. The Rendering Engine interprets the HTML, XML, and JavaScript that comprises a given URL and generates the layout that is displayed in the User Interface. A key component of the Rendering Engine is the HTML parser, this HTML parser is quite complex because it allows the Rendering Engine to display poorly formed HTML pages. 4. The Networking The Networking provides functionality to handle retrieve URLs using the common Internet protocols of HTTP and FTP. The Networking components handles all aspects of Internet communication and security, character set translations and MIME type resolution. The Network component may implement a cache of retrieved documents to minimize network traffic. 5. JavaScript Interpreter The JavaScript Interpreter component executes the JavaScript code that is embedded in a webpage. Results of the execution a passed to the Rendering Engine for display. The Rendering Engine may disable various actions based on user defined properties. 5. The Data Storage The Data Storage manages user data such as bookmarks, cookies and preferences. The new HTML specification (HTML5) defines ‘web database’ which is a complete (although light) database in the browser.  The Rendering Engine:  The responsibility of the rendering engine is well... Rendering, that is display of the requested contents on the browser screen.  By default, the rendering engine can display HTML and XML documents and images. It can display other types of data via plug-ins or extension; for example, displaying PDF documents using a PDF viewer plug-in. However, in this chapter we will focus on the main use case: displaying HTML and images that are formatted using CSS.  Rendering Engine – Flow:  The rendering engine will start getting the contents of the requested document from the networking layer. This will usually be done in 8kB chunks.  Figure 1. The Rendering Engine Flow  After that, this is the basic flow of the rendering engine:  Step 1: Parsing the HTML document and convert elements to DOM nodes in a tree called the “content tree” – HTML Parser  Step 2: Parse the style data, both in external CSS files and in style element together with visual instructions in html will be used to create another tree, call “render tree” – CSS parser  Step 3: After the construction of the render tree it goes through a “layout” process. This means giving each node the exact coordinates where it should appear on the screen.  Step4: The next stage is painting-the render tree will be traversed and each node will be painted using the UI backend layer – Paining  Primary Rendering Engines  Different browsers use different rendering engines: Internet Explorer uses Trident, Firefox uses Gecko, Safari uses WebKit. Chrome and Opera (from version 15) use Blink, a fork of WebKit.  WebKit is an open source rendering engine which started as an engine for the Linux platform and was modified by Apple to support Mac and Windows.   1. **Webkit** 2. **Gecko** 3. **Trident** 4. WebKit Rendering Engine:   WebKit is used as the rendering engine within [Safari](https://en.wikipedia.org/wiki/Safari_(web_browser)) and was formerly used by [Google](https://en.wikipedia.org/wiki/Google)'s [Chrome](https://en.wikipedia.org/wiki/Google_Chrome) web browser on Windows, macOS, and [Android](https://en.wikipedia.org/wiki/Android_(operating_system)) (before version 4.4 KitKat). Chrome used only WebCore, and included its own [JavaScript engine](https://en.wikipedia.org/wiki/JavaScript_engine) named [V8](https://en.wikipedia.org/wiki/Chrome_V8) and a multiprocess system. Chrome for [iOS](https://en.wikipedia.org/wiki/IOS) continues to use WebKit because Apple requires that web browsers on that platform must do so. Other applications on macOS and iOS make use of WebKit, such as Apple's e-mail client [Mail](https://en.wikipedia.org/wiki/Mail_(Apple)), App Store, and the 2008 version of Microsoft's [Entourage](https://en.wikipedia.org/wiki/Microsoft_Entourage) [personal information manager](https://en.wikipedia.org/wiki/Personal_information_manager), both of which make use of WebKit to render HTML content.   * WebKit Embedding API : interface between rendering engine and browser UI. * Webcore: is application logic – loading, parsing, layout, style resolution, painting, event handling, editing, JavaScript bindings. * JSCore (JavaScript Engine) : V8 or JavaScriptCore, parse and executes page logic allows DOM manipulation * Platform API: Network stack, Graphics library, Font engine, Native widgets. * Five ports of WebKit:   Table 1. Five ports of WebKit   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  |  | Chrome  (OS X) | Safari  (OS X) | QtwebKit | Android Browser | Chrome for iOS | | Rendering |  | Skia | CoreGraphics | QtGui | Android stack/Skia | CoreGraphics | | Networking |  | Chromium network stack | CFNetwork | QtNetwork | Font of chromium’s network stack | Chromium stack | | Fonts |  | CoreText via Skia | Core Text | Qt internals | Android stack | CoreText | | JavaScript |  | V8 | JavaSciptCore | JSC (v8 is used elsewhere in Qt | V8 | JavaScriptCore  (without JITting) |   DOM  HTML  HTML Parser  Layout  DOM Tree  Painting  Render tree  Attachment  Style Rules  Style sheet  CSS Parser  Figure 2. The Rendering Flow diagram   1. Gecko Rendering Engine:   Gecko is primarily used in web browsers, the earliest being Netscape 6 and Mozilla Suite (later renamed SeaMonkey). It is also used in other Mozilla web browser derivatives such as [Firefox](https://en.wikipedia.org/wiki/Mozilla_Firefox) and [Firefox for mobile](https://en.wikipedia.org/wiki/Firefox_for_mobile) and the implementation of the [Internet Explorer](https://en.wikipedia.org/wiki/Internet_Explorer)-clone that is part of [Wine](https://en.wikipedia.org/wiki/Wine_(software)). Mozilla also uses it in their [Thunderbird email-client](https://en.wikipedia.org/wiki/Mozilla_Thunderbird).  Gecko is also used by [Sugar](https://en.wikipedia.org/wiki/Sugar_(GUI)) for the [OLPC XO-1](https://en.wikipedia.org/wiki/OLPC_XO-1) computer. Gecko is used as a complete implementation of the [XUL](https://en.wikipedia.org/wiki/XUL) ([XML](https://en.wikipedia.org/wiki/XML) User Interface Language). Gecko currently defines the XUL specification.  Gecko Components   * Document Parser: (HTML and XML Parser) * Style System: contains the CSS Parser and is responsible for getting the CSS data from Necko and parsing it before sending it to the frame constructor * Platform-Specific Rendering and Widgets * Image Library: Interacts with Necko in order to retrieve image data before sending it to the Frame Constructor. * Content Model: Interacts with the various components of Gecko, DOM Storage to gather all the data needed before sending it to the frame constructor. * Frame Constructor: Carries out the task of piece together all the information and actually from the rendering web page before sending it back to the UI through the Platform-Specific Rending subsystem. |
| HTML Parser:  The job of the HTML parser is to parse the HTML markup into a parse tree.  <html>  <body>  <p>  Hello World  </p>  <div> <img src="example.png"/></div>  </body>  </html>    Figure 3. Sample html code  This markup would be translated to the following DOM tree: |

HTMLHtmlElement

HTMLBodyElement

TEXT

HTMLImageElement

HTMLDivElement

HTMLParameterElement

Figure 4. Hierarchical diagram for HTML Parser

CSS Parser:

WebKit uses Flex and Bison parser generators to create parsers automatically from the CSS grammar files. As you recall from the parser introduction, Bison creates a bottom up shift-reduce parser. Firefox uses a top down parser written manually. In both cases each CSS file is parsed into a StyleSheet object. Each object contains CSS rules. The CSS rule objects contain selector and declaration objects and other objects corresponding to CSS grammar.

P,div {

Margin-top: 3px;

}

.error {

Color:red;

}

CSSStyleSheet

CSSRule

CSSRule

Selectors

Selectors

Declarations

Declarations

p

.error

Margin-top

div

3px

color

red

Figure 5. Hierarchical diagram for CSS Parser

The order of processing scripts and style sheets

Script:

The model of the web is synchronous. Authors expect scripts to be parsed and executed immediately when the parser reaches a <script> tag. The parsing of the document halts until the script has been executed. If the script is external then the resource must first be fetched from the network–this is also done synchronously, and parsing halts until the resource is fetched. This was the model for many years and is also specified in HTML4 and 5 specifications. Authors can add the "defer" attribute to a script, in which case it will not halt document parsing and will execute after the document is parsed. HTML5 adds an option to mark the script as asynchronous so it will be parsed and executed by a different thread.

Speculative parsing:

Both WebKit and Firefox do this optimization. While executing scripts, another thread parses the rest of the document and finds out what other resources need to be loaded from the network and loads them. In this way, resources can be loaded on parallel connections and overall speed is improved. Note: the speculative parser only parses references to external resources like external scripts, style sheets and images: it doesn't modify the DOM tree–that is left to the main parser.

Style sheets

Style sheets on the other hand have a different model. Conceptually it seems that since style sheets don't change the DOM tree, there is no reason to wait for them and stop the document parsing. There is an issue, though, of scripts asking for style information during the document parsing stage. If the style is not loaded and parsed yet, the script will get wrong answers and apparently this caused lots of problems. It seems to be an edge case but is quite common. Firefox blocks all scripts when there is a style sheet that is still being loaded and parsed. WebKit blocks scripts only when they try to access certain style properties that may be affected by unloaded style sheets.

Render tree construction

While the DOM tree is being constructed, the browser constructs another tree, the render tree. This tree is of visual elements in the order in which they will be displayed. It is the visual representation of the document. The purpose of this tree is to enable painting the contents in their correct order.

Firefox calls the elements in the render tree "frames". WebKit uses the term renderer or render object.  
A renderer knows how to lay out and paint itself and its children.

Layout

When the renderer is created and added to the tree, it does not have a position and size. Calculating these values is called layout or reflow.

HTML uses a flow based layout model, meaning that most of the time it is possible to compute the geometry in a single pass. Elements later ``in the flow'' typically do not affect the geometry of elements that are earlier ``in the flow'', so layout can proceed left-to-right, top-to-bottom through the document. There are exceptions: for example, HTML tables may require more than one pass ([3.5](https://www.html5rocks.com/en/tutorials/internals/howbrowserswork/#3_5)).

The coordinate system is relative to the root frame. Top and left coordinates are used.

Layout is a recursive process. It begins at the root renderer, which corresponds to the <html> element of the HTML document. Layout continues recursively through some or all of the frame hierarchy, computing geometric information for each renderer that requires it.

The position of the root renderer is 0,0 and its dimensions are the viewport–the visible part of the browser window.

All renderers have a "layout" or "reflow" method, each renderer invokes the layout method of its children that need layout.

The layout process:

The layout usually has the following pattern:

1. Parent renderer determines its own width.
2. Parent goes over children and:
   * 1. Place the child renderer (sets its x and y).
     2. Calls child layout if needed–they are dirty or we are in a global layout, or for some other reason–which calculates the child's height.
3. Parent uses children's accumulative heights and the heights of margins and padding to set its own height–this will be used by the parent renderer's parent.
4. Sets its dirty bit to false.

Firefox uses a "state" object(nsHTMLReflowState) as a parameter to layout (termed "reflow"). Among others the state includes the parents width.  
The output of the Firefox layout is a "metrics" object(nsHTMLReflowMetrics). It will contain the renderer computed height.

Painting

In the painting stage, the render tree is traversed and the renderer's "paint()" method is called to display content on the screen. Painting uses the UI infrastructure component.

CSS2 defines the order of the painting process. This is actually the order in which the elements are stacked in the stacking contexts. This order affects painting since the stacks are painted from back to front. The stacking order of a block renderer is:

* background color
* background image
* border
* children
* outline

### CSS Box model

The CSS box model describes the rectangular boxes that are generated for elements in the document tree and laid out according to the visual formatting model.  
Each box has a content area (e.g. text, an image, etc.) and optional surrounding padding, border, and margin areas.

Figure 6. CSS Box Model

Border

Margin

padding

Content