How modern Web Browsers work[1]

Prof. Dr. Stefan Zander

07. March 2018

> *Objectives:*
>
> - *Learn about the basic building blocks of modern Web browsers*
> - *Get acquainted with the processing internals and the DOM building logic*
> - *Understand what happens inside the browser when you type in an URL*

Contents

1 Preface

> *In the years of IE 90% dominance there was nothing much to do but regard the browser as a "black box", but now, with open source browsers having more than half of the usage share, it's a good time to take a peek under the engine's hood and see what's inside a web browser. Well, what's inside are millions of C++ lines...*
>
> —Tali Garsiel

The following information and facts about the internal operation principles of WebKit and Gecko is the result of extensive research done by the Israeli developer **Tali Garsiel**. Over a few years, she reviewed all the published data about browser internals and spent a lot of time reading Web browser source code. Tali published her research on her site[2]. In the following years, her research results have been revised and republished on numerous occasions and provided insights to a larger audience.

Why should you learn about browser internals?

This lecture note is a revised summary of the excellent article "How Browsers Work: Behind the scenes of modern web browsers" published by Tali Garsiel and Paul Irish in 2011. The original article is available at: https://www.html5rocks.com/en/tutorials/internals/howbrowserswork/.

There is also a video available at vimeo about Tali's talk: http://vimeo.com/44182484.

Learning the internals of browser operations helps you make better decisions and know the justifications behind development best practices. It also helps you to identify performance bottlenecks and build lightning fast websites. As we will see, page loading time has an influence on the Google page rank—a page loading time > 2 sec. results in a lower rank in the Google search results and the Google crawler also crawls such pages less frequently, meaning that search index terms are less frequently updated and the time until new or updated page content will be considered by the Google search engine is extended.

TODO: Add refs

2 Introduction

"I just had to take the hypertext idea and connect it to the TCP and DNS ideas and—ta-da!—the World Wide Web."[3]

—Sir Tim Berners-Lee, inventor of the World Wide Web

[3] One of the famous quotes Sir Tim Berners-Lee used to say about the development of the World Wide Web... [his]

Web browsers are the most widely used software. This lecture explains their fundamental operation principles so that students get an understanding about the things that happen internally when a website is requested, i.e., the time from typing in a website's URL in the browser's address bar until it is rendered by in the browser's viewport.

The complexity of Web browser software has significantly changed over recent years—as the following two screenshots indicate:
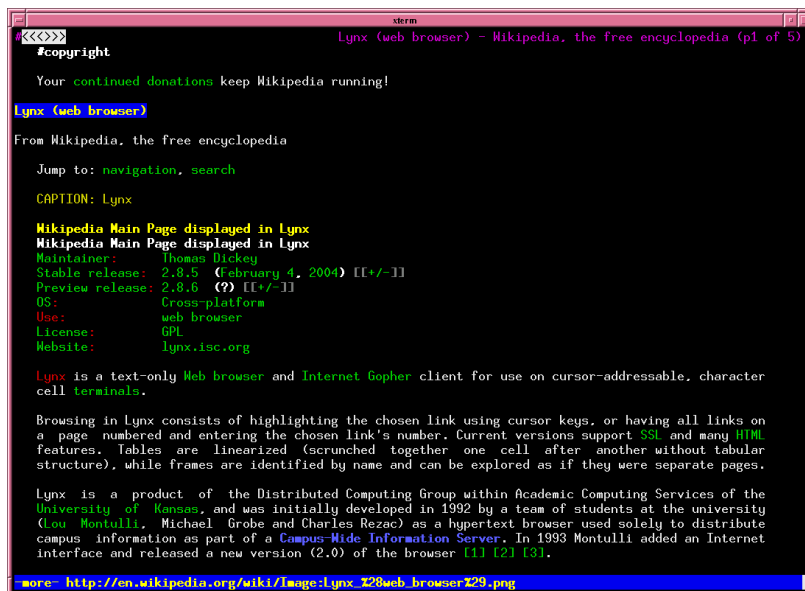

Figure 1: Screenshot of the Lynx Browser, the first and purely text-based browser for the World Wide Web.

TODO: Add image of first NEXT browser; compare it with inspector of Google Chrome; add some description (refs are in the keynote file)
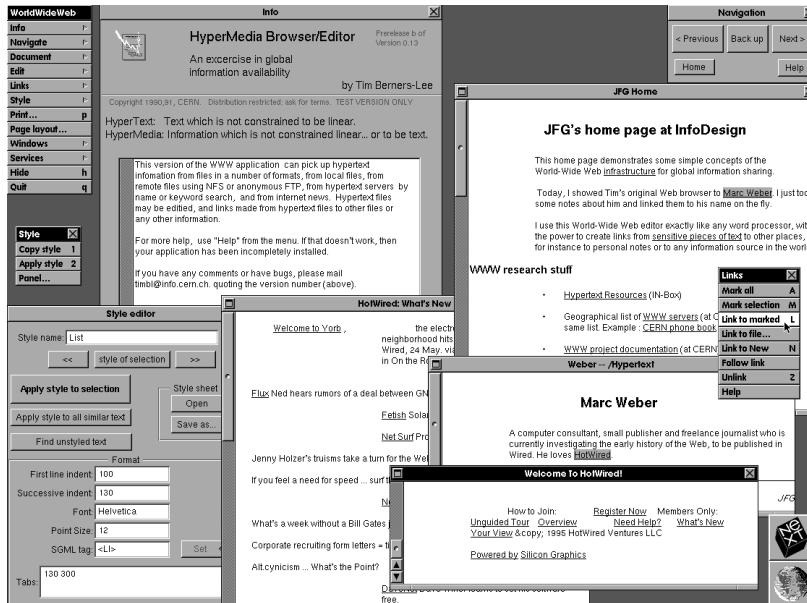
Figure 2: Screenshot of the first WWW browser with a graphical user interface, the NeXT Browser used at CERN.

2.1   What is a Web Browser

This section outlines some of the central functionalities provided by a modern Web browser. It helps in understanding what a browser exactly is.

1. Is an application that we use when we browse the World Wide Web.

2. It renders text-based HTML documents into visual pages, which are what we see inside a browser.

3. It speaks HTTP protocol and communicates with Web servers.

4. It understands URL and knows how to translates URL into Web resources, e.g., HTML text files, images, videos, etc. (*URL dereferencing*)

5. It is a virtual machine that runs the JavaScript programs embedded inside HTML documents.

6. It understands CSS rules and applies the rules to layout the pages.

7. It interacts with a user in front of a browser and translates user inputs into browser events, e.g., clicking a link, clicking a button, submitting text inside a text box.

8. It provides sophisticated tools for analyzing the structure of Web content and network traffic

9. It contains a JavaScript console to utilize its JavaScript engine

3   The Browser's Main Functionality

The main function of a browser is to present the web resource you choose, by requesting it from the server and displaying it in the browser window. The resource is usually an HTML document, but may also be a PDF, image, or some other type of content. The location of the resource is specified by the user using a URI (Uniform Resource Identifier).

The way the browser interprets and displays HTML files is specified in the HTML and CSS specifications. These specifications are maintained by the **World Wide Web Consortium** organization[4], commonly denominated as **W3C**, which is the standards organization for the web. For years browsers conformed to only a part of the specifications and developed their own extensions. That caused serious compatibility issues for web authors. Today most of the browsers more or less conform to the specifications.

[4] https://www.w3.org/

Browser user interfaces have a lot in common with each other. Among the common user interface elements are:

- Address bar for inserting a URI

- Back and forward buttons

- Bookmarking options

- Refresh and stop buttons for refreshing or stopping the loading of current documents

- Home button that takes you to your home page

Strangely enough, the browser's user interface is *not specified* in any formal specification, it just comes from good practices shaped over years of experience and by browsers imitating each other. The HTML5 specification does not define UI elements a browser must have, but lists some common elements. Among those are the address bar, status bar and tool bar. There are, of course, features unique to a specific browser like Firefox's downloads manager.

4   The Browser's High Level Structure

The following components are considered the main building blocks of modern Web browsers (cf. [Grosskurth and Godfrey, 2006]):

A reference architecture for modern Web browsers

1. **The user interface**: this includes the address bar, back/forward button, bookmarking menu, etc. Every part of the browser display except the window where you see the requested page.

2. **The browser engine**: marshals actions between the UI and the rendering engine.

3. **The rendering engine**: responsible for displaying requested content. For example if the requested content is HTML, the rendering engine parses HTML and CSS, and displays the parsed content on the screen.

4. **Networking**: Responsible for network calls such as HTTP requests, using different implementations for different platform behind a platform-independent interface.

5. **UI backend**: used for drawing basic widgets like combo boxes and windows. This backend exposes a generic interface that is not platform specific. Underneath it uses operating system user interface methods.

6. **JavaScript interpreter**: Used to parse and execute JavaScript code.

7. **Data storage**: This is a persistence layer. The browser may need to save all sorts of data locally, such as cookies. Browsers also support storage mechanisms such as localStorage, IndexedDB, WebSQL and FileSystem.

Figure 3 provides a graphical overview of the reference elements together with a visualization of their relationships among each other. Figure 4 shows an implementation of the reference architecture elements for the Firefox browser.
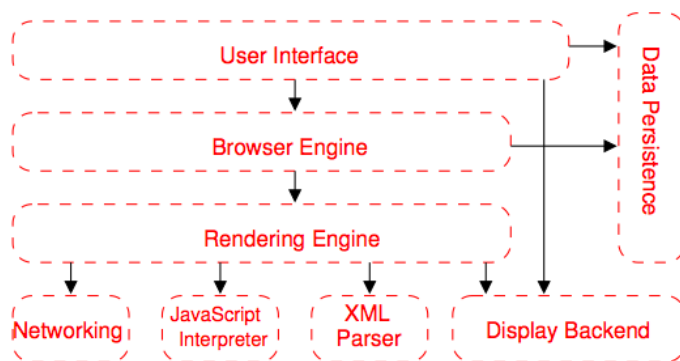

Figure 3: A reference architecture for Web browsers [Grosskurth and Godfrey, 2006]

It is important to note that browsers such as Chrome run multiple instances of the rendering engine: one for each tab. Each tab runs in a separate process.

4.1 The Rendering Engine

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. See webkit.org for more details.
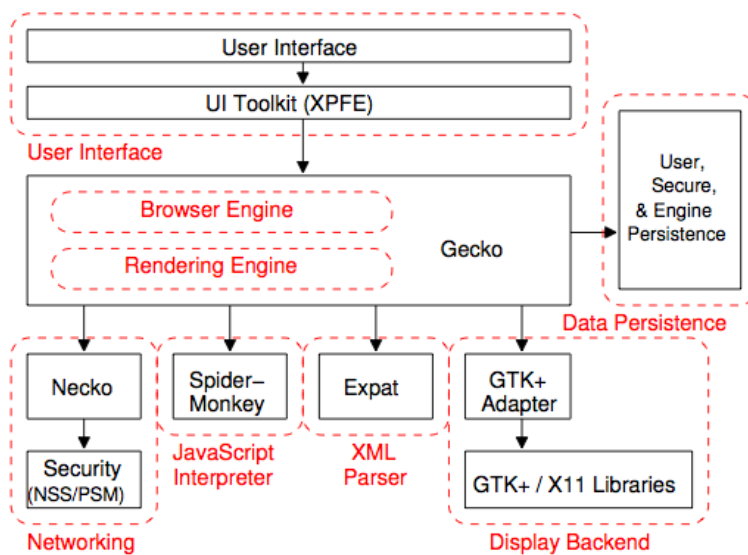
By default the rendering engine can display HTML and XML documents and images. It can display other types of data via plug-ins or extension; for example, displaying PDF documents using a PDF viewer plug-in. However, in this chapter we will focus on the main use case: displaying HTML and images that are formatted using CSS.
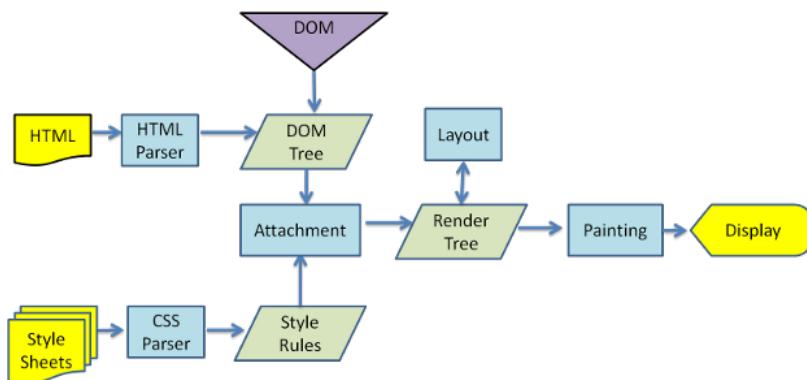
Workflow   The rendering engine will start parsing the HTML document and convert elements to DOM nodes in a tree called the **content tree**. The engine will parse the style data, both in external CSS files and in style elements. Styling information together with visual instructions in the HTML will be used to create another tree: the **render tree**.

The render tree contains rectangles with visual attributes like color and dimensions. The rectangles are in the right order to be displayed on the screen.

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. The next stage is painting—the render tree will be traversed and each node will be painted using the UI backend layer.

It is important to understand that this is a *gradual process*. For better user experience, the rendering engine will try to display contents on the screen as soon as possible[5]. It will not wait until all HTML is parsed before starting to build and layout the render tree. Parts of the content will be parsed and displayed, while the process continues with the rest of the contents that keeps coming from the network.

Parsing   Since parsing is a very significant process within the rendering engine, we will go into it a little more deeply. Let's begin with a little introduction about parsing.

Parsing a document means translating it to a structure the code can use. The result of parsing is usually a tree of nodes that represent the structure of the document. This is called a parse tree or a syntax tree.

Parsing can be separated into two sub processes: **lexical analysis** and **syntax analysis**.

Lexical analysis is the process of breaking the input into tokens. Tokens are the language vocabulary: the collection of valid building blocks. In human language it will consist of all the words that appear in the dictionary for that language.

Syntax analysis is the applying of the language syntax rules.

The parsing process is **iterative**. The parser will usually ask the lexer[6] for a new token and try to match the token with one of the syntax rules. If a rule is matched, a node corresponding to the token will be added to the parse tree and the parser will ask for another token.

If no rule matches, the parser will store the token internally, and keep asking for tokens until a rule matching all the internally stored tokens is found. If no rule is found then the parser will raise an exception. This means the document was not valid and contained syntax errors.

4.2   HTML Content Parsing

There are programs that can create parsers for languages that are defined on the bases of a context free grammar[7,8] by analyzing its grammar, i.e., its vocabulary and syntax rules. Such grammars can usually described using the Backus Naur Form (BNF). However, human language and HTML (as we will see) are not context free grammars. Therefore, a standard parser for parsing HTML does not exist. This has some consequence for Web development:

[5] This is referred to as the **Compositing Forest** (cf. `https://www.chromium.org/developers/design-documents/gpu-accelerated-compositing-in-chrome`) since four conceptually different tree structures are processed in parallel (DOM tree, RenderObject tree, RenderLayer tree, GraphicsLayer tree).

Parsing is based on the syntax rules the document obeys: the language or format it was written in. Every format you can parse must have deterministic grammar consisting of vocabulary and syntax rules. It is called a context free grammar. Human languages are not such languages and therefore cannot be parsed with conventional parsing techniques.

[6] The lexer (sometimes called *tokenizer*) that is responsible for breaking the input into valid tokens.

[7] A language can be parsed by regular parsers if its grammar is a context free grammar.

[8] An intuitive definition of a context free grammar is a grammar that can be entirely expressed in BNF. For a formal definition see your theoretical informatics' lecture notes.

References

The world wide web of tim berners-lee. URL `http://history-computer.com/Internet/Maturing/Lee.html`.

Alan Grosskurth and Michael W. Godfrey. A Reference Architecture For Web Browsers, 2006. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.60.1151`.