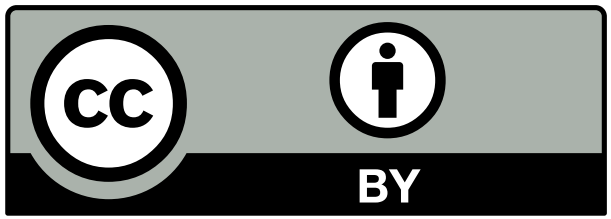
Web Browsers

Jeszenszky Péter

Faculty of Informatics, University of Debrecen

December 9, 2025

Terms of Use



History (1)

The first web browser:

- WorldWideWeb (Tim Berners-Lee, December 25, 1990)
- See:
  - Tim Berners-Lee. The WorldWideWeb browser.
  - Jay Hoffmann. The Web's First (And Second) Browser. September 25, 2017.

History (2)

Further information:

- Jay Hoffmann. The History of the Web.
  - Chapter 2: Browsers
  - Chapter 10: Browser Wars

Recommended Reading about How Browsers Work (1)

- Tali Garsiel, Paul Irish. How Browsers Work: Behind the scenes of modern web browsers. August 5, 2011.
- Anton Paras. Notes on "How Browsers Work". December 11, 2017.
- MDN Web Docs – Populating the page: how browsers work

Recommended Reading about How Browsers Work (2)

- Eric Lawrence (ericlaw). Demystifying Browsers. October 29, 2024.
- Mariko Kosaka. *Inside look at modern web browser (part 1-4)*. 2018.
  - https://developer.chrome.com/blog/inside-browser-part1
  - https://developer.chrome.com/blog/inside-browser-part2
  - https://developer.chrome.com/blog/inside-browser-part3
  - https://developer.chrome.com/blog/inside-browser-part4
- Design Documents (Chromium)
- Pavel Panchekha, Chris Harrelson. Web Browser Engineering. Oxford University Press, 2024.
  - Repository: https://github.com/browserengineering/book

Structure of a Web Browser

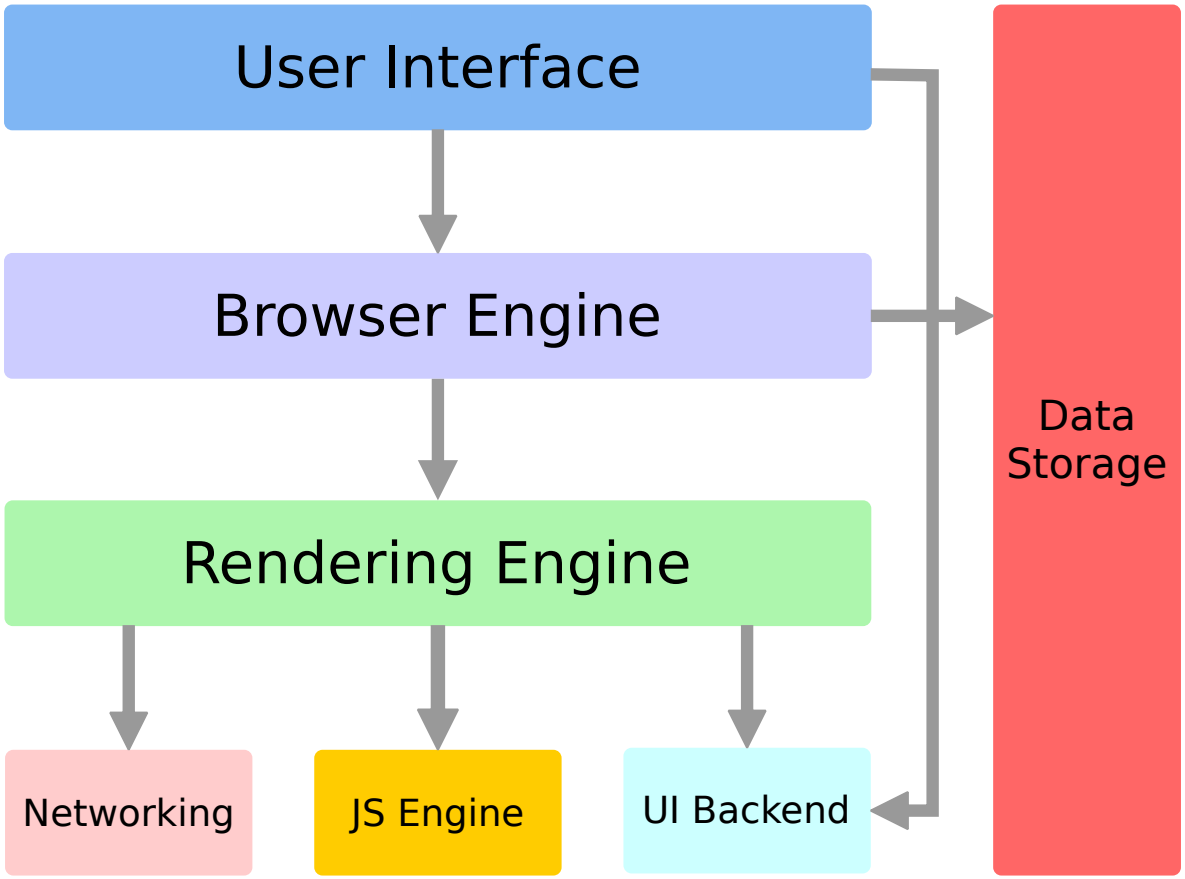


Figure 1: Source: https://web.dev/articles/howbrowserswork

Browser Engine vs Rendering Engine

- The terms “browser engine”, “rendering engine”, and “layout engine” are often used interchangeably, although it is technically inaccurate.
- Gecko is a special case, since it consists of a browser engine and a rendering engine.

Components of Web Browsers: User Interface

- The user interface includes the address bar, buttons (e.g., Back, Forward, Refresh), menus, etc.
- It includes every visible part of the web browser except the area in which the web page is displayed.

Components of Web Browsers: Browser Engine

- Provides a high-level interface for querying and manipulating the rendering engine.
- Acts as a middleman between the user interface and the rendering engine.
- Communicates with the data storage component.

Components of Web Browsers: Rendering Engine

- A core component of web browsers.
- Its function is to create a visual representation of web content (e.g, an HTML or XML document) to be displayed on an output device.
- The rendering of content happens according to the formatting rules specified.
- Browsers can run a separate instance of the rendering engine for each tab.
    - For example, Chromium and Google Chrome work this way.

Components of Web Browsers: Networking Component (1)

- The networking component is responsible for handling network communication, including:
  - HTTP requests
  - WebSocket API
  - WebRTC
- It provides a platform-independent interface, behind which platform-specific APIs are used.

Components of Web Browsers: Networking Component (2)

Further information:

- WebSockets: WebSockets Living Standard (WHATWG)
- WebRTC:
  - https://webrtc.org/
  - WebRTC: Real-Time Communication in Browsers. W3C Recommendation 13, March 2025.

Components of Web Browsers: JavaScript Engine (1)

- The JavaScript engine is responsible for executing JavaScript code.
- First, JavaScript code is compiled into bytecode.
- Then, bytecode is compiled into machine code using just-in-time (JIT) compilation.

Components of Web Browsers: JavaScript Engine (2)

Major JavaScript engines:

- SpiderMonkey (written in: C++/Rust/JavaScript; license: Mozilla Public License 2.0) https://spidermonkey.dev/
  - The JavaScript engine of the Mozilla Project.
- V8 (written in: C++; license: New BSD License) https://v8.dev/ https://chromium.googlesource.com/v8/v8.git
  - The JavaScript engine of Chromium.
- JavaScriptCore (written in: C++; license: LGPLv2) https://developer.apple.com/documentation/javascriptcore https://trac.webkit.org/wiki/JavaScriptCore
  - The JavaScript engine of Apple.

Components of Web Browsers: JavaScript Engine (3)

- Other JavaScript engines:
  - GraalVM JavaScript (written in: C/C++; license: Universal Permissive License v1.0) https://www.graalvm.org/latest/reference-manual/js/ https://github.com/oracle/graaljs
    - JavaScript engine to execute JavaScript and Node.js applications in GraalVM.
- For more, see: A list of JavaScript engines, runtimes, interpreters

Components of Web Browsers: UI Back-end

- The UI back-end is responsible for drawing in the web browser.
- It is used both for displaying user interface elements and the web page.
- It provides a platform-independent interface, behind which platform-specific APIs are used.
- To improve performance, it may use the GPU.

Components of Web Browsers: Data Storage (1)

Data storage is responsible for persisting data locally; it is used for:

- HTTP cookies
- HTTP caching
- Indexed Database API (IndexedDB)
- Web Storage API

Components of Web Browsers: Data Storage (2)

Further information:

- HTTP cookies:
  - Adam Barth. HTTP State Management Mechanism. RFC 6265, April 2011.
- HTTP caching:
  - Roy T. Fielding (ed.), Mark Nottingham (ed.), Julian F. Reschke (ed.). HTTP Caching. RFC 9111, June 2022.
- Indexed Database API (IndexedDB):
  - Indexed Database API 3.0. W3C Working Draft, 13 August 2025.
- Web Storage API:
  - HTML Living Standard – Web storage

Difficulties of Rendering Web Pages

- Users' expectations:
  - Web pages should load fast
  - Interaction with the page should be smooth
- Technical difficulties:
  - Network latency
  - Rendering a web page requires many additional resources, i.e., images, stylesheets, and scripts
  - For the most part, browsers are single-threaded

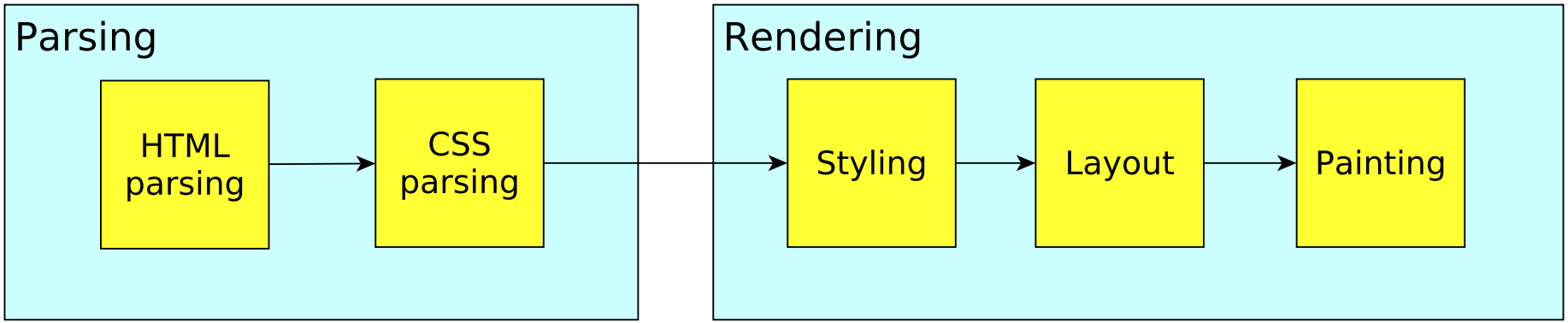
Steps of Rendering a Web Page

- Navigation: the user enters a URI into the address bar
- A DNS lookup is made to determine the IP address of the server.
  - Note that additional DNS lookups may be necessary to render the page.
- A network (e.g., TCP) connection is established to the server.
- The browser sends an initial HTTP request, to which the server sends back an HTTP response that, in most cases, contains an HTML representation.
- The process continues with the steps of the critical rendering path.
- Other tasks that are not discussed: JavaScript compilation, building the accessibility tree.

Critical Rendering Path (1)

- The critical rendering path is the sequence of steps the browser goes through to render an HTML or XML document on an output device.
- See:
  - Populating the page: how browsers work (MDN)
  - Critical rendering path (MDN)

Critical Rendering Path (2)



Critical Rendering Path (3)

1. **HTML parsing**: DOM tree construction
2. **CSS parsing**: CSSOM tree construction
3. **Styling**: render tree (a.k.a., "box tree") construction
   - Elements that aren't going to be displayed (e.g., `title` or `script`) are not included in the render tree.
4. **Layout**: determining the sizes and positions of each node in the render tree
   - The first time the size and position of each node is determined is called layout.
   - Subsequent recalculations of the sizes and positions are called reflows.
   - For replaced elements (e.g., images) for which the dimensions are not known, a placeholder space is allocated initially. A reflow happens when the dimensions are determined.
5. **Paiting**: painting the individual nodes of the render tree on the output device
   - Performance can be improved by involving the GPU.

Resource Loading in Browsers (1)

- By default, parsing and executing JavaScript code blocks the parsing of an HTML document.
  - However, `script` elements with the `async` or the `defer` attribute are treated differently.

Resource Loading in Browsers (2)

- Parsing CSS stylesheets does not block parsing HTML; browsers download them asynchronously.
  - However, it blocks rendering.
- Downloading images, audio, or video files does not block parsing HTML; browsers can download them asynchronously.
  - However, the `<img>` element can take the `lazy` attribute that affects when the browser downloads the image.
  - Similarly, the `<audio>` and `<video>` elements can take the `preload` attribute, which also affects how the resources are downloaded.

Error Handling in HTML (1)

- Unlike previous versions of HTML, the current specification defines in some detail the required processing for invalid documents as well as valid documents.
- Web browsers are very tolerant of errors; they automatically fix invalid content.
  - However, this applies to documents in the HTML syntax!
- See:
  - HTML Standard – Parsing HTML documents
    - An introduction to error handling and strange cases in the parser

Error Handling in HTML (2)

- Trying out the examples:
  - Examine the DOM trees in either the browser DevTools or the Live DOM Viewer!

Error Handling in HTML (3)

Example:

```
<html>
<body><b>How is <i>it</b> rendered?</i></body>
</html>
```

The invalid document is parsed into a DOM tree that is the same as that of the following valid document:

```
<html>
<head></head>
<body><b>How is <i>it</i></b><i> rendered?</i></body>
</html>
```

Error Handling in HTML (4)

Example:

```
<html>
  <foo>
  <p bar class=>Does it <b>work,<i> or</b> not</i>?
  <
  </body>
</html>
```

Error Handling in CSS (1)

See:

- CSS Syntax Module Level 3. W3C Candidate Recommendation Draft, 24 December 2021.
  - For a high-level overview of error handling, see subsection 2.2. Error Handling.
- CSS error handling (MDN)

Error Handling in CSS (2)

Example: The style rule

```
p {
  color: white
  background color: navy;
  font-family: sans-serif;
  unknown: 0;
  font-size: xxx;
}
```

is effectively the same as the following style rule:

```
p {
  font-family: sans-serif;
}
```

Major Rendering Engines

- WebKit
- Blink
- Gecko

Rendering Engines: WebKit

- Website: https://webkit.org/
- Repository: https://github.com/WebKit/WebKit
- Developer: Apple
- License: LGPLv2/Simplified BSD License
- Written in: C++
- WebKit-based software:
  - Safari
  - Google Chrome for iOS
  - Firefox for iOS
  - GNOME Web

Rendering Engines: Blink

- Website: https://www.chromium.org/blink/
- Repository: https://chromium.googlesource.com/chromium/blink/
- Developer: Chromium Project (Google)
- License: Simplified BSD License/New BSD License/LGPLv2.1
- Written in: C++
- Originally, Blink was born as a fork of the WebCore component of WebKit.
- Blink-based software:
  - Chromium
  - Google Chrome, Google Chrome for Android
  - Chromium-based Microsoft Edge
  - Opera
  - Vivaldi

Rendering Engines: Gecko

- Website: https://wiki.mozilla.org/Gecko:Home_Page
- Repository: https://hg.mozilla.org/mozilla-central/
- Developer: Mozilla Project
- License: Mozilla Public License 2.0
- Written in: C, C++, JavaScript, Rust
- Gecko-based software:
  - Firefox
  - Firefox for Android
  - SeaMonkey

Market Share of Browsers

- All platforms: https://gs.statcounter.com/browser-market-share
- Desktop: https://gs.statcounter.com/browser-market-share/desktop
- Mobile: https://gs.statcounter.com/browser-market-share/mobile

Major Desktop Browsers

- Chromium, Google Chrome
- Firefox
- Opera
- Safari
- Microsoft Edge
- Brave

Comparison of Web Browsers

- Browser comparison (Can I use...)
- Comparison of Web Browsers (eylenburg.github.io)
- PrivacyTests.org
- Compare Firefox with other browsers (Mozilla)

Technical Documentation for Developers

- Chromium, Google Chrome: For Developers
- Firefox:
  - https://developer.mozilla.org/en-US/docs/Mozilla/Firefox
  - https://firefox-source-docs.mozilla.org/
- Microsoft Edge: Microsoft Edge for Developers

Major Desktop Browsers: Chromium

- Website: https://www.chromium.org/Home/
- Repository: https://chromium.googlesource.com/chromium/src.git
- Developer: Chromium Project (Google)
- License: New BSD License
  - See: https://chromium.googlesource.com/chromium/src/+/master/LICENSE
- Written in: C, C++, JavaScript
- Platform: Linux, macOS, Windows
- Rendering engine: Blink
- JavaScript engine: V8

Major Desktop Browsers: Google Chrome

- Website: https://www.google.com/chrome/
- Developer: Google
- License: proprietary (see: `chrome://terms`)
- Written in: C, C++, JavaScript
- Platform: Linux, macOS, Windows
- Rendering engine: Blink
- JavaScript engine: V8
- Technical information: `chrome://system`
- Developer edition: Google Chrome for developers

Major Desktop Browsers: Chromium and Google Chrome (1)

Google Chrome is based on Chromium.

Major Desktop Browsers: Chromium and Google Chrome (2)

- Differences between the two browsers:
  - They have different logos.
  - They are equipped with a different set of audio and video codecs.
  - User data are stored stored in different directories.
    - Chromium: `~/.cache/chromium`, `~/.config/chromium`
    - Google Chrome: `~/.cache/google-chrome`, `~/.config/googlechrome`
  - Operating system distributions may modify the source code of Chromium; packaging also depends on the distribution.
- See: The Difference between Google Chrome and Chromium on Linux

Major Desktop Browsers: Firefox

- Website: https://www.mozilla.org/firefox/
- Repository: https://hg.mozilla.org/mozilla-central/
  https://searchfox.org/mozilla-central/source
- Developer: Mozilla Project
- License: Mozilla Public License 2.0 (see: `about:license`)
- Written in: C, C++, JavaScript, Rust
- Platform: Linux, macOS, Windows
- Rendering engine: Gecko
- JavaScript engine: SpiderMonkey
- System information: `about:support`
- Developer edition: Firefox Developer Edition

Major Desktop Browsers: Opera

- Website: https://www.opera.com/
- Developer: Opera Software
- License: proprietary
  - See: https://legal.opera.com/eula/computers/
- Written in: C++
- Platform: Linux, macOS, Windows
- Rendering engine: Blink
- JavaScript engine: V8
- Developer edition: Opera Developer

Major Desktop Browsers: Safari

- Website: https://www.apple.com/safari/
- Developer: Apple
- License: proprietary
- Written in: C++, Objective-C
- Platform: macOS
- Rendering engine: WebKit
- JavaScript engine: JavaScriptCore
- Developer edition: Safari Technology Preview

Major Desktop Browsers: Microsoft Edge (1)

- Website: https://www.microsoft.com/edge
- Developer: Microsoft
- License: proprietary (see: `edge://terms`)
- Platform: Linux (2020-), macOS (2019-), Windows
- Rendering engine: EdgeHTML (-2019), Blink (2019-)
- JavaScript engine: Chakra (-2019), V8 (2019-)

Major Desktop Browsers: Microsoft Edge (2)

Chromium-based Edge:

- Announcement:
  - Joe Belfiore. Introducing Microsoft Edge Beta: Be one of the first to try it now. August 20, 2019.
- Written in: C, C++, JavaScript

Major Desktop Browsers: Brave

- Website: https://brave.com/
- Repository: https://github.com/brave/brave-browser
- Developer: Brave Software
- License: Mozilla Public License 2.0 (see: <brave://terms>)
- Written in: C++, JavaScript, Rust
- Platform: Linux, macOS, Windows
- Rendering engine: Blink
- JavaScript engine: V8

Web Developer Tools

- Chromium, Google Chrome, Opera: Chrome DevTools
- Firefox: Firefox DevTools User Docs
- Safari: Safari Developer Help
- Chromium-based Edge: Microsoft Edge DevTools documentation

Browser Add-ons

- A browser add-on is a software module for customizing the appearance or behavior of a web browser.
- Types of add-ons: extensions, language packs, themes, plugins

Browser Extensions

- A browser extension is a browser add-on that alters the behavior of a web browser, i.e., adds new functionality or modifies existing functionality.

Available Browser Add-ons

- Chromium, Google Chrome: Chrome Web Store
- Firefox: Firefox Add-ons
- Opera: Opera add-ons
- Safari: Mac App Store
- Chromium-based Edge: Microsoft Edge Addons

Standardization Efforts on Browser Extensions

WebExtensions Community Group (W3C)

- Website: https://www.w3.org/community/webextensions/
- Repository: https://github.com/w3c/webextensions/
- Initiators of the group: Apple, Google, Microsoft, Mozilla
- The goal of the community group is to specify and standardize a common browser extension platform that enables developers not to have to rewrite their extensions completely to work in different browsers.

Developing Browser Extensions (1)

- Chromium, Google Chrome, Chromium-based Edge:
  - See: https://developer.chrome.com/docs/extensions
- Firefox:
  - Extensions for Firefox are built using the WebExtensions API, a cross-browser system for developing extensions.
  - This technology is, to a large extent, compatible with the extension API supported by Chromium-based browsers. In most cases, extensions written for Chromium-based browsers run in Firefox with just a few changes.
  - See: Browser Extensions (MDN)

Developing Browser Extensions (2)

- Opera:
  - See: Extensions Documentation
- Safari:
  - Extensions for the Safari browser must be created with the Xcode IDE.
  - See: Safari Extensions

Developing Browser Extensions (3)

- Extensions are built on web technologies, such as HTML, CSS, and JavaScript.
- In the case of the major browsers discussed, they are bundles of HTML, CSS, JavaScript, and other files (e.g., images).
- Chromium, Google Chrome, Firefox, Opera, Microsoft Edge:
  - Every extension has a manifest file (`manifest.json`) that contains metadata and ties together the components.

Developing Browser Extensions (4)

Further information:

- Firefox Extension Workshop

Headless Browsers (1)

- A headless browser is a web browser without a graphical user interface.
- It can be controlled programmatically.
- Possible applications:
  - Automated testing of web applications
  - Automating interaction with web pages
  - Taking screenshots of web pages
  - Web scraping
  - ...
- See: https://en.wikipedia.org/wiki/Headless_browser

Headless Browsers (2)

The Chromium, Google Chrome, and Firefox browsers can run in headless mode.

- Chromium, Google Chrome:
  - Mathias Bynens, Peter Kvitek. Chrome Headless mode.
  - https://chromium.googlesource.com/chromium/src/+/lkgr/headless/README.md
- Firefox:
  - Using Headless Mode in Firefox

Headless Browsers (3)

Taking a screenshot where output is written to the file `screenshot.png`:

- Google Chrome:

```
google-chrome --headless --screenshot \
--window-size=1920,1080 https://www.w3.org/
```

- Firefox:

```
firefox -screenshot --window-size=1920,1080 \
https://www.w3.org/
```

Headless Browsers (4)

Free and open source headless browsers:

- HtmlUnit (written in: Java; license: Apache License 2.0)
  https://www.htmlunit.org/ https://github.com/HtmlUnit/htmlunit
  - Provides a way to simulate a browser for testing purposes and is intended to be used within another testing framework, such as JUnit.
- Puppeteer (written in: JavaScript; license: Apache License 2.0)
  https://pptr.dev/ https://github.com/puppeteer/puppeteer
  - JavaScript library that provides a high-level API to control Chrome or Firefox.

See: dhamaniasad/HeadlessBrowsers: A list of (almost) all headless web browsers in existence

Text-based Web Browsers

- Lynx (platform: Linux, macOS, Windows; license: GPLv2)
  https://lynx.invisible-island.net/
  - The oldest web browser that is still in general use and is being actively developed.
- Links (platform: Linux, macOS, Windows; license: GPLv2)
  http://links.twibright.com/
  - It can also run in graphics mode, see the –g command line option.
- Browsh (platform: Linux, macOS, Windows; license: LGPLv2.1)
  https://www.brow.sh/ https://github.com/browsh-org/browsh
  - Creates a purely text-based version of web pages using Firefox running in headless mode.

Major Mobile Browsers

- Google Chrome (Android, iOS)
- Safari for iOS
- Samsung Internet for Android
- UC Browser (Android, iOS)

Mobile Browsers for iOS

- Web browsers for iOS must use the WebKit rendering engine and the JavaScriptCore engine.
- See: App Store Review Guidelines – Software Requirements
  Apps that browse the web must use the appropriate WebKit framework and WebKit JavaScript.

Major Mobile Browsers: Chrome

- Website: https://www.google.com/chrome/
- Developer: Google
- License: proprietary
- Platform: Android, iOS
- Rendering engine: Blink (Android)/iOS WebKit (iOS)
- JavaScript engine: V8 (Android)/JavaScriptCore (iOS)

Major Mobile Browsers: Safari for iOS

- Website: https://www.apple.com/safari/
- Developer: Apple
- License: proprietary
- Rendering engine: iOS WebKit
- JavaScript engine: JavaScriptCore

Major Mobile Browsers: Samsung Internet for Android

- Website: https://www.samsung.com/us/support/owners/app/samsung-internet
- Developer: Samsung
- License: proprietary
- Rendering engine: Blink
- JavaScript engine: V8
- Further information for developers: https://developer.samsung.com/internet

Major Mobile Browsers: UC Browser

- Website: https://www.ucweb.com/
- Developer: UCWeb, a subsidiary of Alibaba
- License: proprietary
- Platform: Android, iOS
- Rendering engine: Blink (Android)/iOS WebKit (iOS)
- JavaScript engine: V8 (Android)/JavaScriptCore (iOS)

Major Mobile Browsers: Opera

- Opera for Android/iOS, Opera Mini (platform: Android), Opera GX (platform: Android, iOS)
- Website: https://www.opera.com/
- Developer: Opera Software
- License: proprietary
- Rendering engine: Blink (Android)/iOS WebKit (iOS)
- JavaScript engine: V8 (Android)/JavaScriptCore (iOS)

The about URI Scheme (1)

- Web browsers widely use the `about` URI to designate access to their internal resources, such as settings, application information, or hidden built-in functionality (e.g., Easter eggs).
  - See: S. Moonesamy (ed.). The "about" URI Scheme. RFC 6694, August 2012.
- For example, the `about:blank` URI references a blank page.

The about URI Scheme (2)

- Some browsers map about URIs to their own equivalents, replacing the scheme name (i.e., `about`) with their name (such as `chrome`, `opera`, or `edge`).
  - For example, Chromium and Google Chrome use the URI `chrome://about` instead of `about:about`.
- An exception is `about:blank`, which is left unchanged.

The about URI Scheme (3)

Browser support:

- Chromium, Google Chrome:
  - `about:` is mapped to `chrome://`
  - The list of Chrome URIs: `chrome://about`
  - Examples: `chrome://bookmarks`, `chrome://dino`, `chrome://flags/`, `chrome://history`, ...
    - For debugging purposes: `chrome://crash`, `chrome://quit`, `chrome://restart`, ...
- Firefox:
  - The list of about URIs: `about:about`
  - Examples: `about:config`, `about:downloads`, `about:rights`, ...

The about URI Scheme (4)

Browser support: (continued)

- Opera: Supports a subset of `chrome` URIs and a few additional ones, but the scheme name is replaced with `opera`.
  - Examples: `opera://downloads`, `opera://emoji-picker`, `opera://flags`, `opera://history`, ...
- Safari: Only `about:blank` is recognized.

The about URI Scheme (5)

Browser support: (continued)

- Chromium-based Edge:
  - `about:` is mapped to `edge://`
  - The list of Edge URIs: `edge://about`
  - Examples: `edge://downloads`, `edge://flags`, `edge://history/all`, `edge://history/today`, `edge://history/yesterday`, ...

Support for Web Technologies in Browsers

- Current support:
  - Can I use... Support tables for HTML5, CSS3, etc
  - Chrome Platform Status
- Roadmap:
  - Mozilla Specification Positions
  - WebKit Standards Positions

Rendering Modes of Browsers (1)

Layout engines render HTML documents in the following modes:

- Quirks mode
- Standards mode
- Almost standards modes

Rendering Modes of Browsers (2)

- The historical reason for the existence of rendering modes is that early web standards were not compatible with the behavior of web browsers existing at that time.
- Web browsers introduced a new rendering mode to comply with web standards while still being able to render existing older existing content properly.
- Thus, modern web pages conforming to current web standards and obsolete web pages are rendered differently.

Rendering Modes of Browsers (3)

- **Quirks mode**: mimicking (emulating) the behavior of legacy web browsers in a way that violates current web standards for rendering obsolete web pages.
- **Standards mode**: rendering web pages according to current web standards.
- **Almost standards mode**: some web browsers also have a third rendering mode that differs from standards mode only in some height calculations that affects, e.g., the layout of images inside table cells.

Rendering Modes of Browsers (4)

- Earlier, browser engines used slightly different quirks modes, however, the WHATWG has standardized the quirks mode in the Quirks Mode specification.
- This specification does not enumerate all quirks that currently exist in browsers, a number of quirks are specified in other WHATWG specifications.

Rendering Modes of Browsers (5)

- The DOM specification renames standards mode and almost standards mode:
  - Renames standards mode to **no-quirks mode**.
  - Renames almost standards mode to **limited-quirks mode**.

Rendering Modes of Browsers (6)

Determining which rendering mode to use for an HTML document:

- For documents transmitted with the `text/html` media type the document type declaration determines the rendering mode.
- Documents transmitted with the `application/xhtml+xml` media type are always rendered in standards mode.
  - In this case the document is parsed with an XML parser that also checks the document for well-formedness.

Rendering Modes of Browsers (7)

How to determine which rendering mode is used?

- The `compatMode` attribute of the `Document` DOM interface returns the string `"BackCompat"` if the document's rendering mode is quirks, and `"CSS1Compat"` otherwise.
  - See: DOM – Living Standard – Interface Document
- On the browser UI:
  - Firefox: Firefox Page Info window (available from the Tools menu)

Rendering Modes of Browsers (8)

Further useful links:

- MDN Web Docs – Quirks Mode and Standards Mode
- Henri Sivonen. Activating Browser Modes with Doctype.