Advancing WebDriver BiDi Support in WebKit

Lauro Moura (lmoura@igalia.com)

Selenium Conference 2025
2025-03-27

About me

- At Igalia since 2020
- WebKit team
  - QA, WebDriver
- Other work done in the past:
  - EFL (JS and C# bindings)
  - Maemo (Python bindings, Nix WebKit port)

About Igalia

- Specialized **Open Source consultancy**, founded in 2001
- **Fully remote**, HQ in **A Coruña** (Spain). **Flat structure**.
- **Top contributors** to all the main **Web Rendering Engines**
  - WebKit, Chromium, Gecko and Servo
- **Active contributor to other areas and OSS projects**
  - V8, SpiderMonkey, JSC, LLVM, Node.js, GStreamer, Mesa, Linux Kernel...
- **Members of several working groups**:
  - W3C, WHATWG, WPT, TC39, OpenJS, Test262, Khronos...

Agenda

- Overview of WebKit
- WebDriver (Classic) in WebKit
- Adding WebDriver BiDi support
- Main challenges
- Next Steps

WebKit

WebKit != Safari

Why Browser Engines ≠ Real Desktop Browsers...

David Burns, BrowserStack @ Selenium Conference 2023

WebKit

- **Open Source** Web Rendering **Engine**
  - Started as a fork of KHTML and KJS in 2001
  - Opensourced by Apple in 2005
  - Chrome's *Blink* forked out of WebKit in 2013
- Support for **different platforms**:
  - **Desktop & Mobile**: Mac, iOS and Linux
  - **Embedded**: set-top-boxes, video game consoles, smart home appliances, infotainment, digital signage...

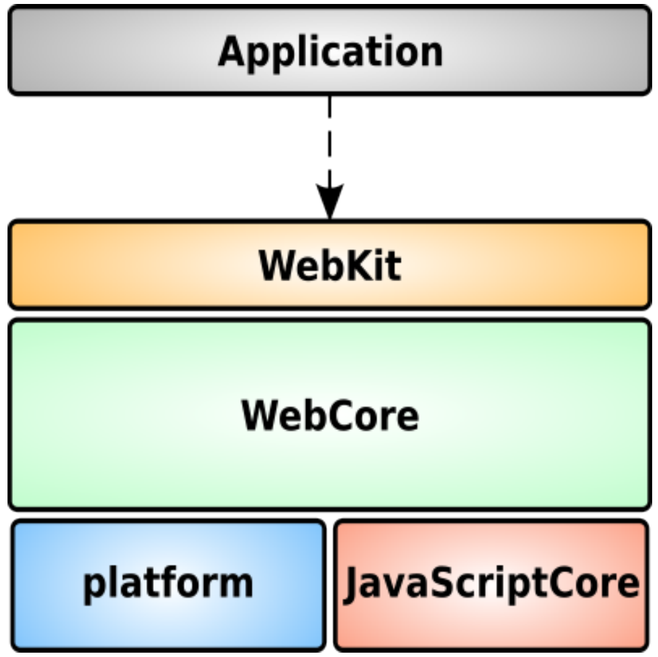
Who uses WebKit?

- **Safari** uses WebKit
- **Mail** uses WebKit
- **GNOME Web (Epiphany)** uses WebKit
- **Evolution** uses WebKit
- **Playstation** uses WebKit
- **Your set-top-box** might be using WebKit
- Even **kitchen mixers** can use WebKit

WebKit, really

WebKit Architecture

- **Application**:
  - What end-users interact with
- **WebKit**:
  - Exposes an API to applications and implements the split-process model
- **WebCore**:
  - Layout, rendering, network, multimedia, accessibility…
- **JavaScriptCore**:
  - The JavaScript engine
- **Platform**:
  - Platform-specific hooks



WebKit Portability

- WebKit is a **cross-platform** engine
- But some operations are **system-specific...**
  - Networking, graphics, multimedia, user input
- Or the system might use **specific languages or frameworks** in its API
  - Different UI toolkits

WebKit *Ports*

- Implementation of low level operations
  - Using system-specific libraries
  - Core APIs, GLIB, GStreamer, etc.
- Expose a Developer API, wrapping WebKit's internal API. For example:
  - OS X framework on Mac
  - GObject-based API on Linux

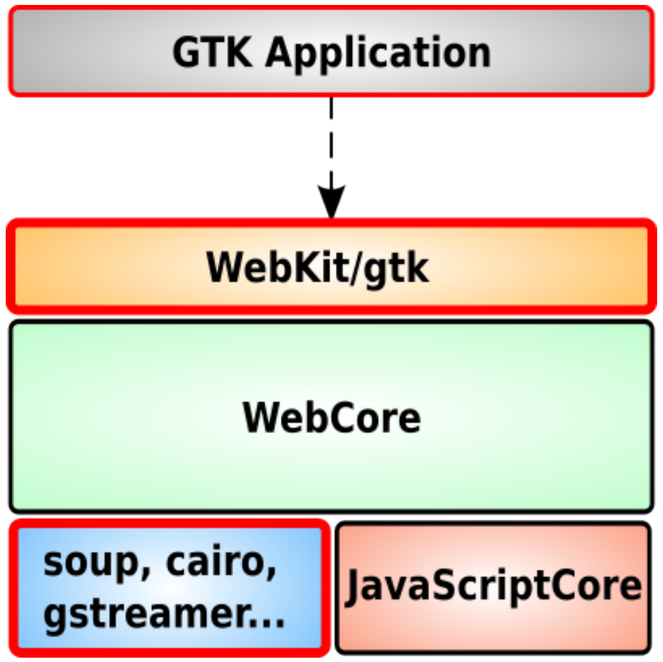
WebKit *Ports*

- Adaptation to a **specific platform/API**
- Official **upstream** WebKit ports:
  - **Mac**: Safari, Apple Mail, iTunes, App Store
  - **iOS**: Ditto on iOS devices
  - **WinCairo**: Microsoft Playwright, Playstation SDK
  - **Playstation**: Playstation 4 and 5
  - **WebKitGTK**: GNOME Web (Née Epiphany), Evolution
  - **WPEWebKit**: Custom made embedded browsers (e.g. set-top-boxes interfaces/players)

https://docs.webkit.org/Ports/Introduction.html

WebKit Ports: WebKitGTK example

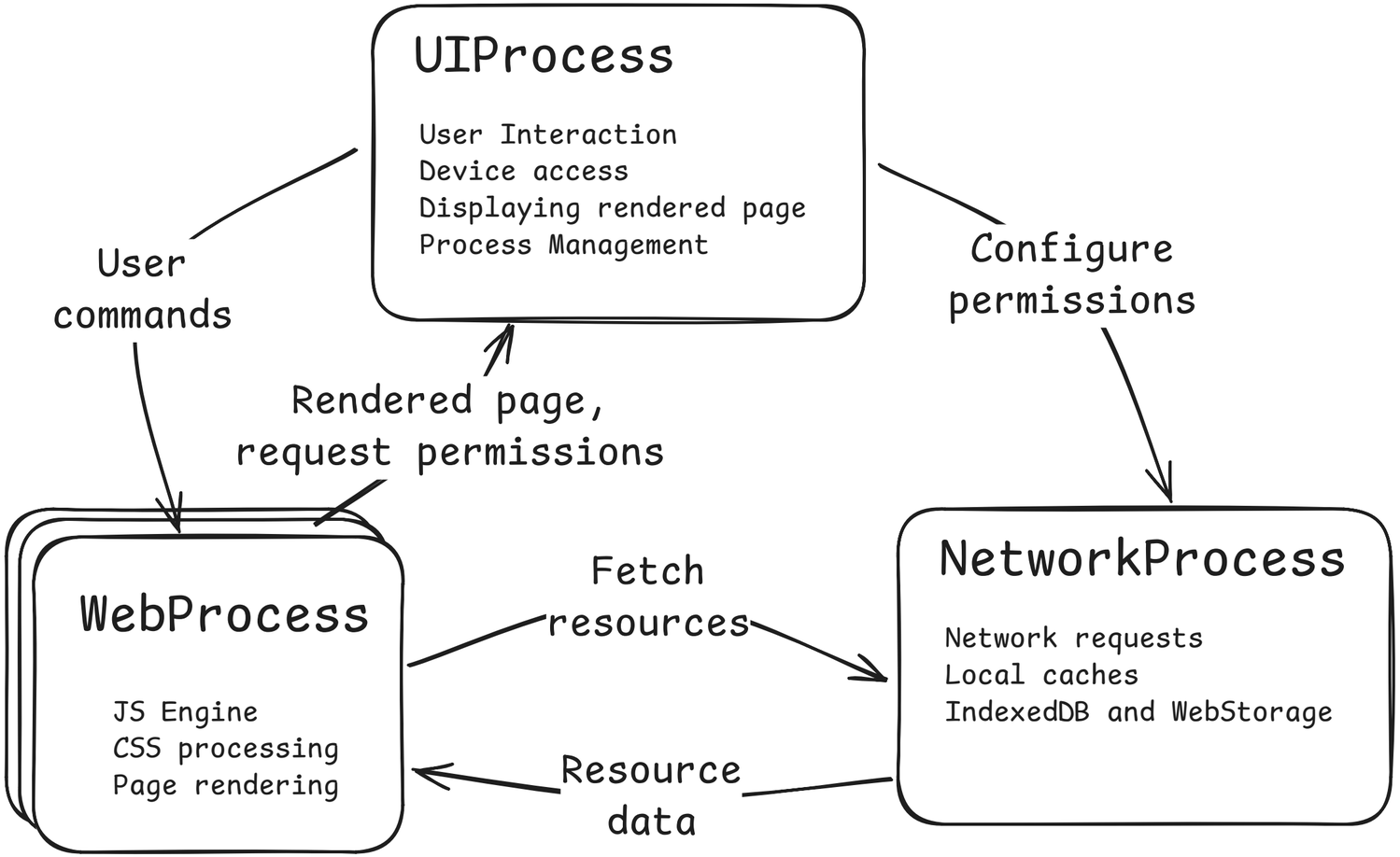
- Networking: **libsoup**
- Graphics: **GTK** (widgets) + **Skia** (low level)
- Multimedia: **GStreamer**
- API: **GObject**-based



WebKit: Multi-process architecture

- **UIProcess**: Browser executable
  - User interaction, display rendered content, process management
- **WebProcess**: Web content
  - Rendering, JS execution, DOM trees handling
- **NetworkProcess**: Network and storage
  - Browsing session
  - Network requests, caches, `IndexedDB`, `WebStorage`
- Communication through **platform-specific IPC** messages

WebKit: Multi-process architecture



WebDriver (Classic) in WebKit

WebDriver Classic in WebKit

- **2016: WebDriver support in Safari 10**
  - **W3C's WebDriver** was under heavy development
- Implementation split between the **WebKit** (the library) and the **Drivers**
  - Support can differ between ports...
- Some features may **ask the browser** to do the work
  - For example: Creating new windows/tabs

Two WebKit drivers

- `WebKitWebDriver` - **Open source**, inside WebKit's repository
  - `Source/WebDriver` directory
  - Used by **WebKitGTK**, **WPEWebKit**, **WinCairo**
- `safaridriver` - **Proprietary**, focus on automating **Safari**
- But both use **the same protocol** to talk to the browser
- *Note*: Unless specified otherwise, we'll focus on the open source driver

`WebKitWebDriver`

- Runs the WebDriver Classic's **HTTP server**
- Spawns and manages the **browser instances**
  - Or connect to an already running browser
- **Initial processing** of most commands
  - Capabilities negotiation, dialog handling, navigation waiting
- Uses the *async* **Web Inspector** protocol to talk to the Browser
- Currently, limited to **1 active session** per driver instance

WebDriver Classic: Browser side

- `WebAutomationSession` is the main entry point on the `UIProcess`
  - Dispatches the actual commands **inside the browser**
  - e.g. **Synthesizing input events** using the port's input API
- But some commands need to **touch Web content** directly
  - e.g. **executing JS** or getting information about **DOM elements**
  - `WebAutomationSessionProxy` handles this on the `WebProcess` side

WebDriver testing in WebKit

- **Imported test suites** into the WebKit repository
  - **Selenium** python tests (`py/`)
  - W3C **Web Platform Tests** (`webdriver/`)
- **Long interval** between updates in the past :(
  - Sometimes over a year
  - Led to **issues** in WebKitGTK and WPEWebKit wrappers in Selenium
  - Planning **more frequent** updates
- `MiniBrowser` as the test browser
- **Post-commit CI bots**
  - Running against **WebKitGTK** and **WPEWebKit**
  - Keeping the test suite green is **challenging**

wpt.fyi

Path	Chrome 136 Linux 20.04 02db8c4 Mar 22, 2025	Firefox 138 Linux 20.04 02db8c4 Mar 22, 2025	Safari 215 preview macOS 15.3 02db8c4 Mar 22, 2025	WebKitGTK 2.49 Linux 20.04 02db8c4 Mar 23, 2025
bidi/	4389 / 4470	4265 / 4476	1 / 4482	31 / 4482
classic/	3270 / 3455	3428 / 3455	1443 / 2084	2448 / 3438
interop/	13 / 22	22 / 22	0 / 22	0 / 22
Subtest Total	7672 / 7947	7715 / 7953	1444 / 6588	2479 / 7942

webdriver/tests dashboard

WebDriver Classic in WebKit: Limitations

- Stateful commands (`current browsing context`) influenced driver design
- At most **1 session per driver**
- "Known elements" tracking bugs
  - Returning "stale element reference" instead of "no such element"

WebDriver Classic shortcomings

- But the **main limitation** comes from the **protocol itself**
- **The browser can't push events to the driver /o**
- **Polling** is unreliable and might **not cover all use cases**
- We need a **bidi**rectional solution...

WebDriver BiDi in WebKit

WebDriver BiDi 101

- A **W3C** standard **under development** by the **Browser Testing and Tools WG**
  - **Editor's draft** available at **w3c.github.io/webdriver-bidi**
  - Advancing into **Working Draft** status
  - Already **good support** in Chrome and Firefox
- BiDi stands for **Bi-Di**rectional
- Extends **WebDriver Classic**, inspired by **CDP (Chrome DevTools Protocol)**
- **Asynchronous** commands by design
- The browser can **push events** to the driver \o/

BiDi in WebKit - First steps

- Originally developed in a separate branch, but **merged into WebKit's mainline** last year
- Initial work done on the **WebKitGTK/WPEWebKit** Linux ports
- Adding support for the **WebSocket server** (using **libsoup**) and **session creation**
- Basic **log** message **events**
- Using **Seleniun Python** and WPT's `webdriver` libraries to validate the implementation

BiDi in WebKit - Scaling up

- Initially, **handwritten parsing** of incoming messages on the driver
  - This would not scale well with BiDi's extensive data types
- **@burg** added support to **define the commands API declaratively** on the browser side
- **Automatic parsing** of the types defined in the BiDi spec
  - Allows focusing on the implementation of the actual command steps
- **Sharing more code** between the different ports
- We are **polishing this scheme** to follow with implementing the new BiDi commands and event system

BiDi in WebKit: Supported features

- Mixed Classic/BiDi session creation
  - 1 session per driver instance
- `log.entryAdded` with *text* payload for `console` and JS exceptions
  - Missing extra log parameters and information like stack trace, source information

Demo

BiDi in WebKit - Challenges

Multiple active browsing contexts

- Most BiDi commands can run against any given browsing context
- Some existing code still assume we have one active browsing context at a time
  - Mostly on the `Source/WebDriver` side

Multiple user contexts

- A BiDi *user context* is like a browser "profile", with separate storage
- The spec allows running some commands or subscribing to events on separate user contexts
- Currently, `MiniBrowser` (the test browser) lacks this support
- Will require additions to the developer-facing API, requesting the browser to create new profiles

BiDi's event system

- BiDi events are very powerful
- Subscriptions across different *browsing contexts* and *user contexts*
- Event subscription priorities
- The current **Inspector protocol** between the driver and the browser might not be enough
  - Moving event subscription inside the browser, proxied by the driver

WebDriver in WebKit - Next Steps

Next Steps - BiDi

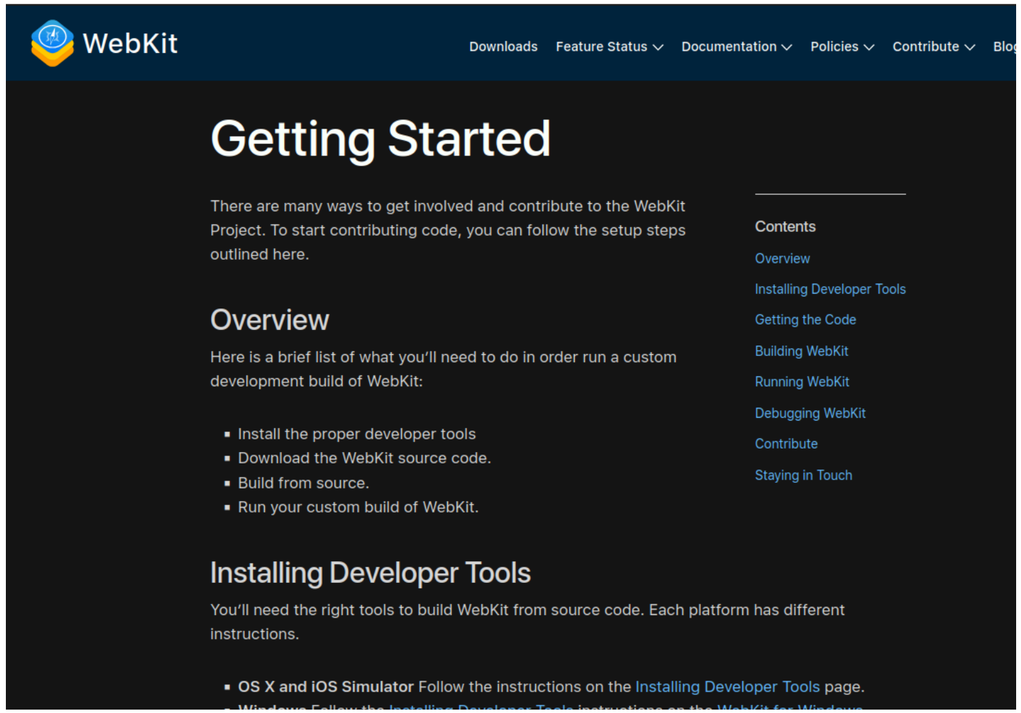
- Implement **basic BiDi commands** required by **WPT** tests
  - `browsingContext.getTree`
  - `browsingContext.navigate`
  - `script.evaluate`
- Support **Selenium Python** BiDi features
  - Finish `log.entryAdded` implementation
  - Keep track of new commands and events supported
- Other sources to help prioritize the next commands:
  - BiDi features in **other Selenium languages** (Preloading scripts? Networking?)
  - **Puppeteer BiDi support**
  - Other low hanging fruits in BiDi's **spec** and **roadmap**

Next Steps - WebDriver QA

- Garden the **current failures**
  - A green tree will help add WebDriver to **pre-commit CI**
- Work **closer** with the testing **community**
  - Update the test suite **more frequently**
  - Ensure **upstream** tooling works with **newer** WebKit
- **Keep WebDriver Classic working!**

Contributing to WebKit

Contributing to WebKit



WebKit

Downloads Feature Status Documentation Policies Contribute Blog

Getting Started

There are many ways to get involved and contribute to the WebKit Project. To start contributing code, you can follow the setup steps outlined here.

Contents

Overview

Here is a brief list of what you'll need to do in order run a custom development build of WebKit:

- Install the proper developer tools
- Download the WebKit source code.
- Build from source.
- Run your custom build of WebKit.

Installing Developer Tools

You'll need the right tools to build WebKit from source code. Each platform has different instructions.

- **OS X and iOS Simulator** Follow the instructions on the Installing Developer Tools page.

https://webkit.org/getting-started/

Triaging and gardening

- **190 open bugs in WebDriver component** in WebKit's Bugzilla
  - Some with incomplete information
  - **BiDi support meta bug**
- **WebDriver builders** in WebKit's Buildbot are not green
  - Triaging existing failures
  - Especially after updating the test suites

Recap

- WebKit is a **browser engine** used in **many platforms** and use cases, including Safari
- Currently, WebKit **has support** for simple BiDi's `log.entryAdded` **events**
- **Working closer with the community** will be important to prioritize features and ensure the implementation is sound
- BiDi work in WebKit is picking up pace, **so stay tuned!**
  - **Web Engines Hackfest** (Coruña, Spain + Remote, June 2-4)
  - **WebKit Contributors Meeting** (Bay Area, U.S., 2025 H2)

Questions?

Thank you!

igalia