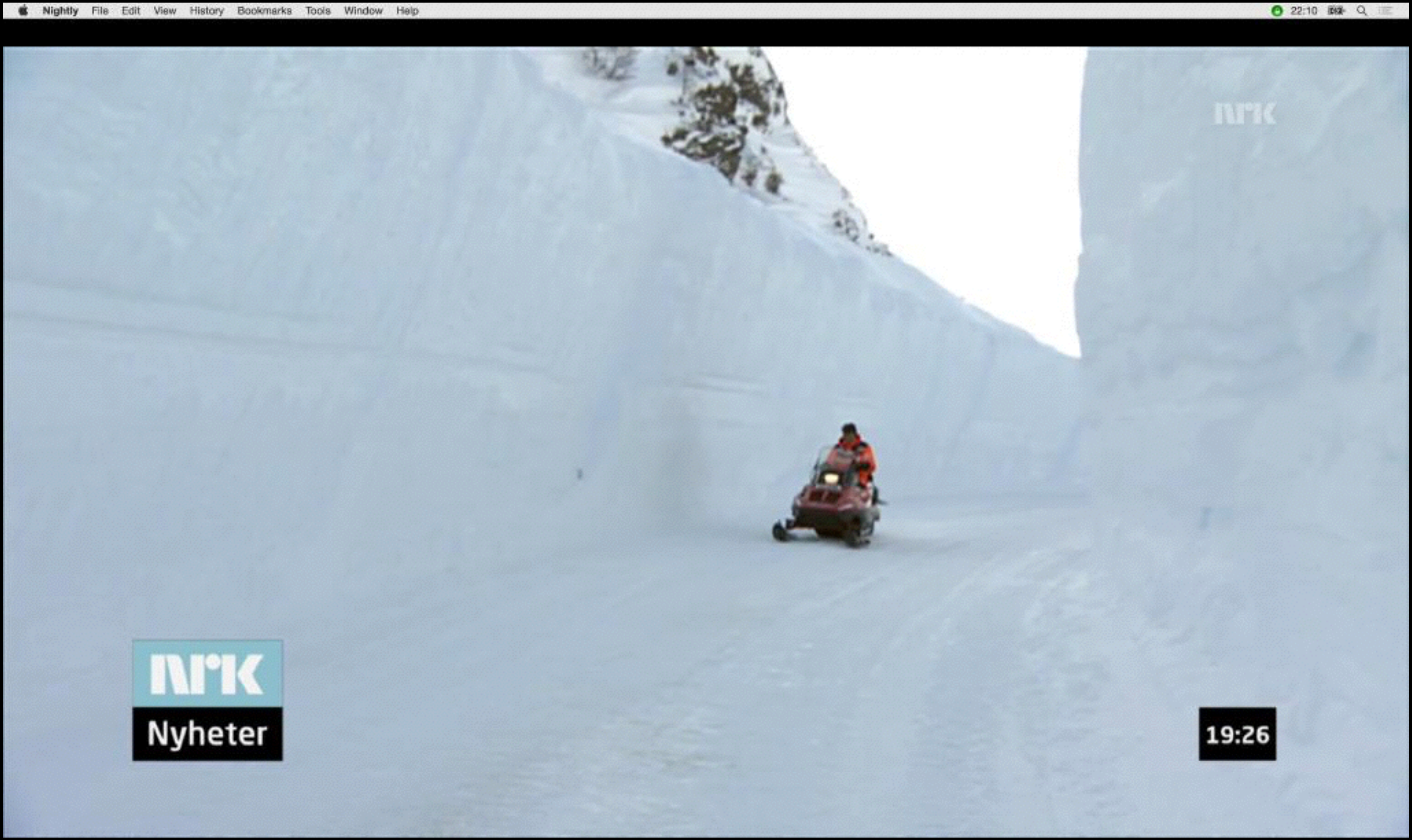
The Future of WebDriver

Andreas Tolfsen
ato@mozilla.com
@tolfsen


Nightly File Edit View History Bookmarks Tools Window Help
22:10
NRK
NRK
Nyheter
19:26

Toolsmith at Mozilla

Co-author of specification

Implementor

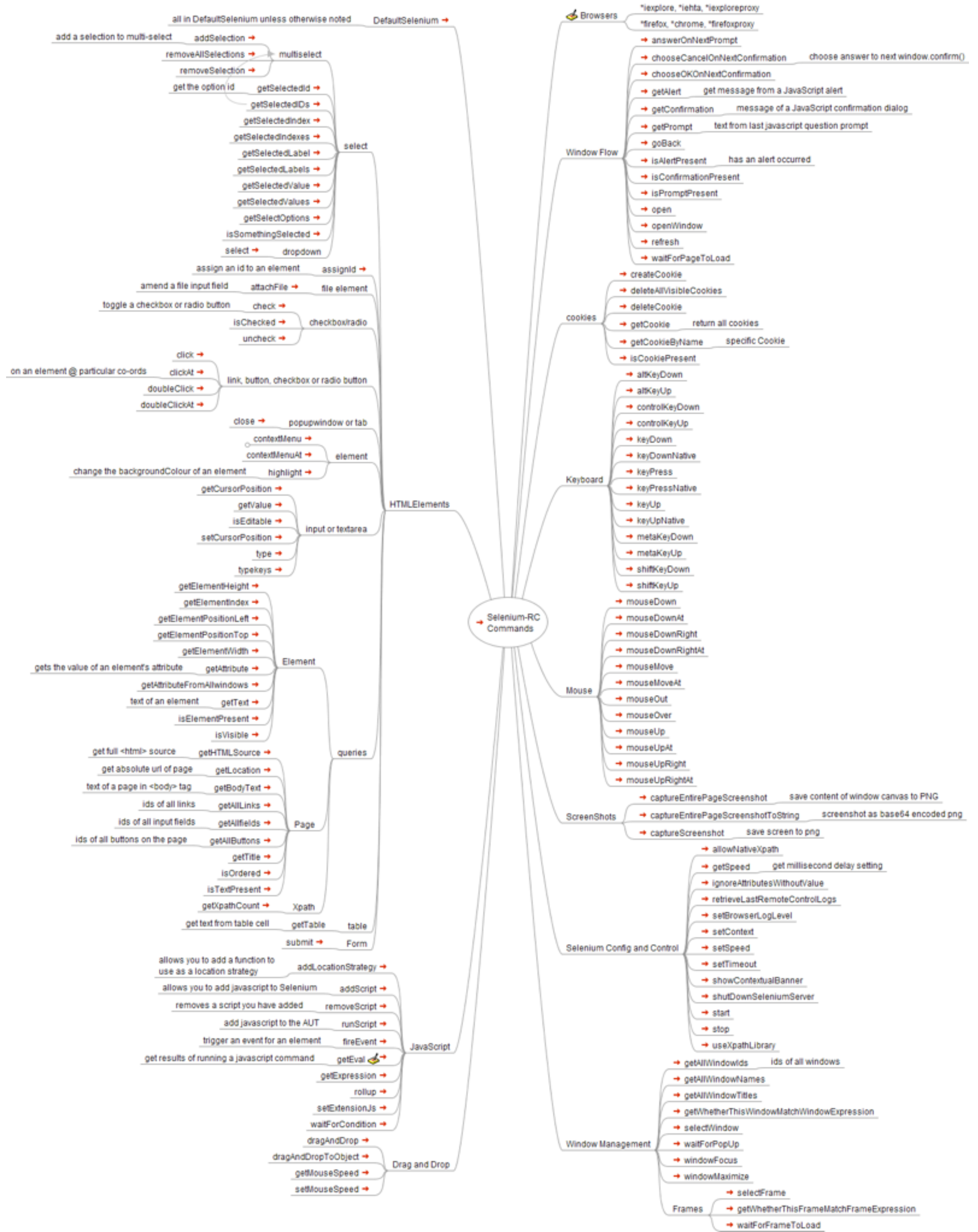
Core committer on Selenium

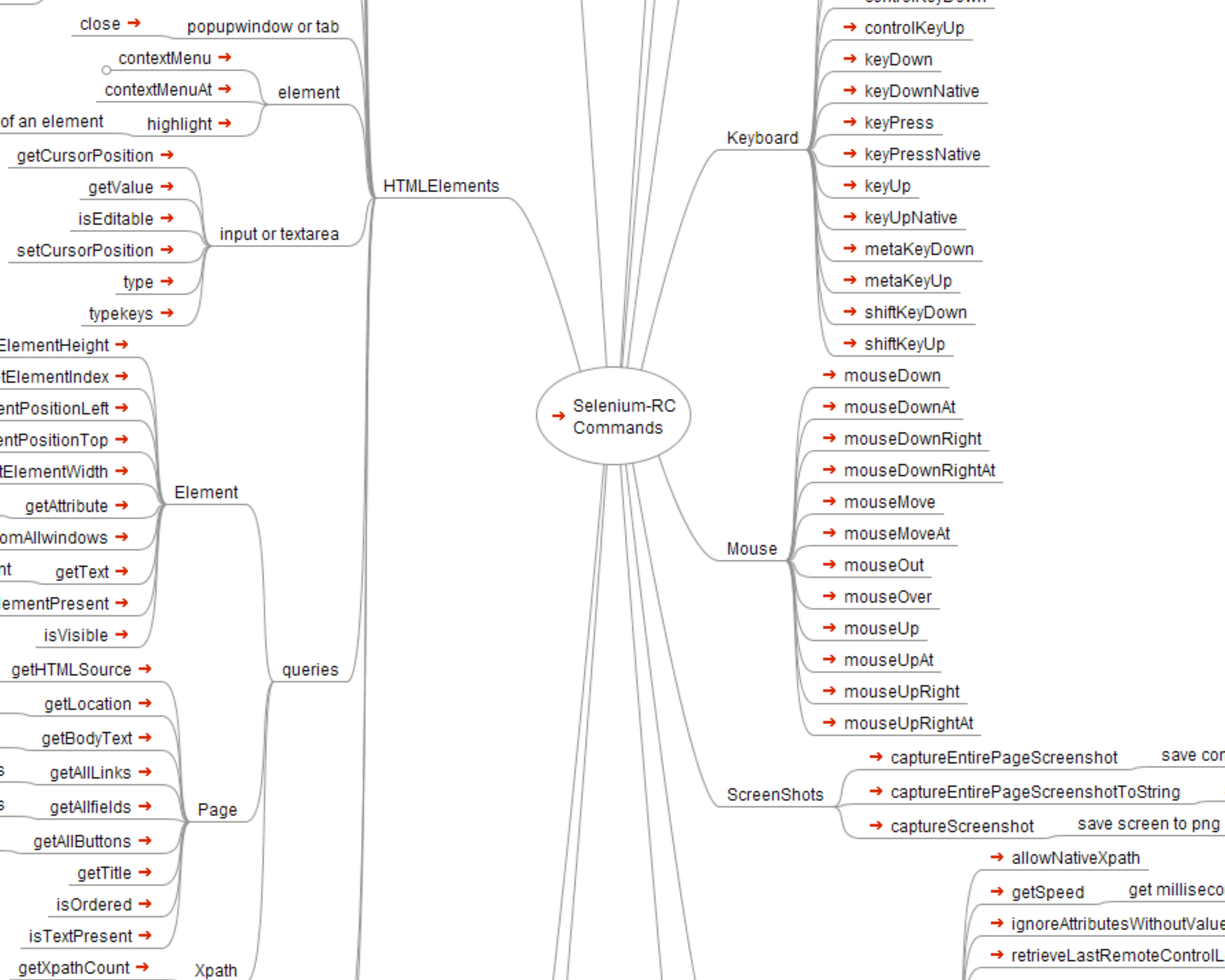
SELENIUM

Umbrella project for tools and libraries related to browser automation

HISTORY

- Selenium RC came out of Thoughtworks
- WebDriver happened at Google
- The big merge
- Selenium 2.0 released 7 July 2011
    - Was a drop-in replacement for Selenium RC


Selenium-RC Commands
DefaultSelenium
all in DefaultSelenium unless otherwise noted
multiselect
addSelection
add a selection to multi-select
removeAllSelections
removeSelection
select
getSelectedId
get the option id
getSelectedIDs
getSelectedIndex
getSelectedIndexes
getSelectedLabel
getSelectedLabels
getSelectedValue
getSelectedValues
getSelectOptions
isSomethingSelected
dropdown
select
HTMLElements
assignId
assign an id to an element
file element
attachFile
amend a file input field
checkbox/radio
check
toggle a checkbox or radio button
isChecked
uncheck
link, button, checkbox or radio button
click
clickAt
on an element @ particular co-ords
doubleClick
doubleClickAt
popupwindow or tab
close
element
contextMenu
contextMenuAt
highlight
change the backgroundColour of an element
input or textarea
getCursorPosition
getValue
isEditable
setCursorPosition
type
typekeys
queries
Element
getElementHeight
getElementIndex
getElementPositionLeft
getElementPositionTop
getElementWidth
getAttribute
gets the value of an element's attribute
getAttributeFromAllwindows
getText
text of an element
isElementPresent
isVisible
Page
getHTMLSource
get full <html> source
getLocation
get absolute url of page
getBodyText
text of a page in <body> tag
getAllLinks
ids of all links
getAllfields
ids of all input fields
getAllButtons
ids of all buttons on the page
getTitle
isOrdered
isTextPresent
Xpath
getXpathCount
table
getTable
get text from table cell
Form
submit
JavaScript
addLocationStrategy
allows you to add a function to use as a location strategy
addScript
allows you to add javascript to Selenium
removeScript
removes a script you have added
runScript
add javascript to the AUT
fireEvent
trigger an event for an element
getEval
get results of running a javascript command
getExpression
rollup
setExtensionJs
waitForCondition
Drag and Drop
dragAndDrop
dragAndDropToObject
getMouseSpeed
setMouseSpeed
Browsers
*iexplore, *iehta, *iexploreproxy
*firefox, *chrome, *firefoxproxy
Window Flow
answerOnNextPrompt
chooseCancelOnNextConfirmation
choose answer to next window.confirm()
chooseOKOnNextConfirmation
getAlert
get message from a JavaScript alert
getConfirmation
message of a JavaScript confirmation dialog
getPrompt
text from last javascript question prompt
goBack
isAlertPresent
has an alert occurred
isConfirmationPresent
isPromptPresent
open
openWindow
refresh
waitForPageToLoad
cookies
createCookie
deleteAllVisibleCookies
deleteCookie
getCookie
return all cookies
getCookieByName
specific Cookie
isCookiePresent
Keyboard
altKeyDown
altKeyUp
controlKeyDown
controlKeyUp
keyDown
keyDownNative
keyPress
keyPressNative
keyUp
keyUpNative
metaKeyDown
metaKeyUp
shiftKeyDown
shiftKeyUp
Mouse
mouseDown
mouseDownAt
mouseDownRight
mouseDownRightAt
mouseMove
mouseMoveAt
mouseOut
mouseOver
mouseUp
mouseUpAt
mouseUpRight
mouseUpRightAt
ScreenShots
captureEntirePageScreenshot
save content of window canvas to PNG
captureEntirePageScreenshotToString
screenshot as base64 encoded png
captureScreenshot
save screen to png
Selenium Config and Control
allowNativeXpath
getSpeed
get millisecond delay setting
ignoreAttributesWithoutValue
retrieveLastRemoteControlLogs
setBrowserLogLevel
setContext
setSpeed
setTimeout
showContextualBanner
shutDownSeleniumServer
start
stop
useXpathLibrary
Window Management
getAllWindowIds
ids of all windows
getAllWindowNames
getAllWindowTitles
getWhetherThisWindowMatchWindowExpression
selectWindow
waitForPopUp
windowFocus
windowMaximize
Frames
selectFrame
getWhetherThisFrameMatchFrameExpression
waitForFrameToLoad


close
popupwindow or tab
contextMenu
contextMenuAt
element
of an element
highlight
getCursorPosition
getValue
isEditable
input or textarea
setCursorPosition
type
typekeys
HTMLElements
ElementHeight
tElementIndex
entPositionLeft
entPositionTop
tElementWidth
getAttribute
Element
omAllwindows
nt
getText
ementPresent
isVisible
queries
getHTMLSource
getLocation
getBodyText
s
getAllLinks
s
getAllfields
getAllButtons
Page
getTitle
isOrdered
isTextPresent
getXpathCount
Xpath
Selenium-RC Commands
controlKeyUp
keyDown
keyDownNative
keyPress
Keyboard
keyPressNative
keyUp
keyUpNative
metaKeyDown
metaKeyUp
shiftKeyDown
shiftKeyUp
mouseDown
mouseDownAt
mouseDownRight
mouseDownRightAt
mouseMove
mouseMoveAt
Mouse
mouseOut
mouseOver
mouseUp
mouseUpAt
mouseUpRight
mouseUpRightAt
captureEntirePageScreenshot
save cor
ScreenShots
captureEntirePageScreenshotToString
captureScreenshot
save screen to png
allowNativeXpath
getSpeed
get milliseco
ignoreAttributesWithoutValue
retrieveLastRemoteControlL

WEBDRIVER API

- WebDriver's **API** is simple and concise by design
- Two main concepts:
  - *WebDriver* represents the browser
  - *WebElement* represents a DOM element
- Designed to be ubiquitous in its application
  (It's a library, not a testing framework!)

mozilla

MISCONCEPTIONS

- Myth: WebDriver is object-oriented and RC isn't
- Distinction between **API** and **SPI** (primitives/service provider interface)

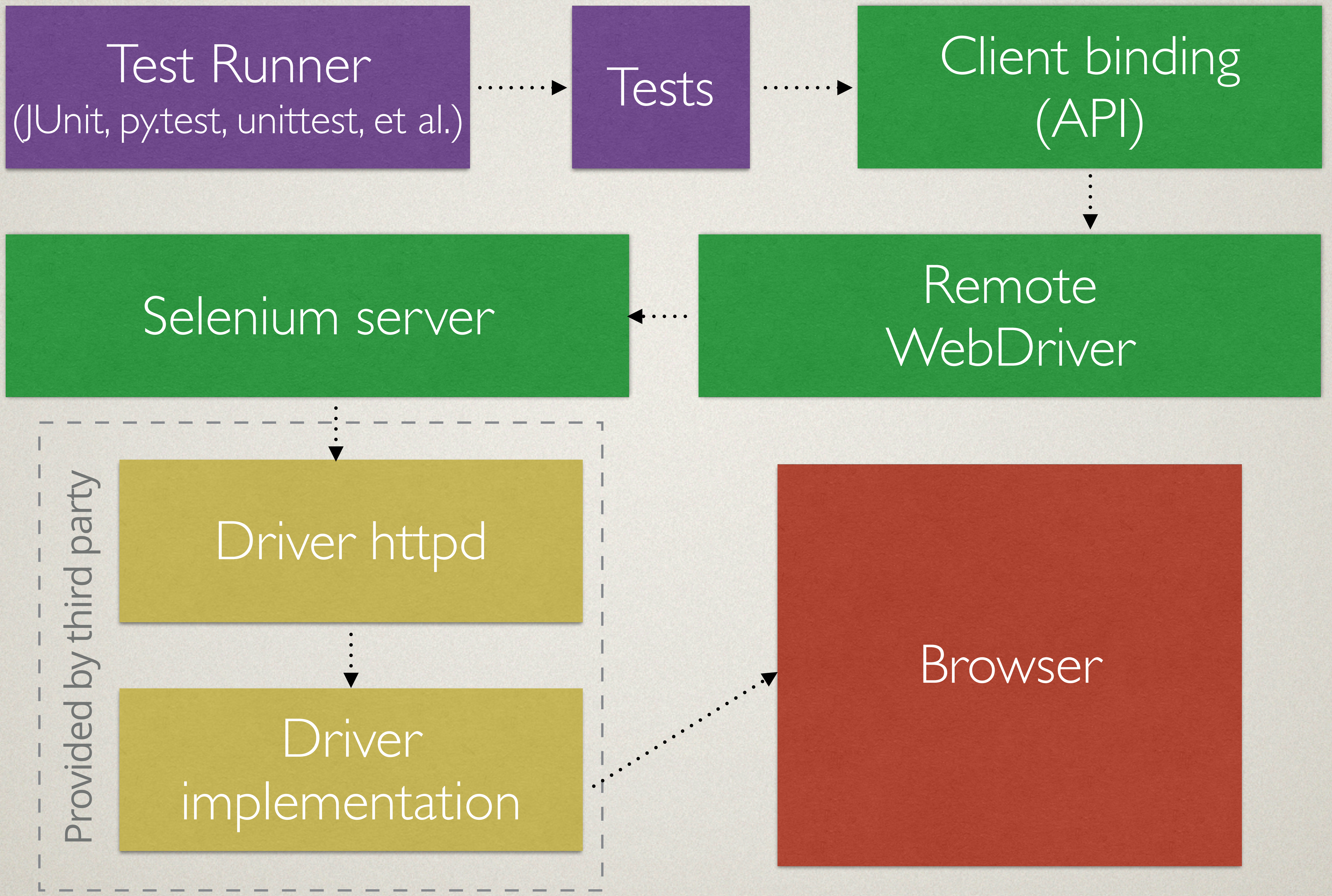
API:

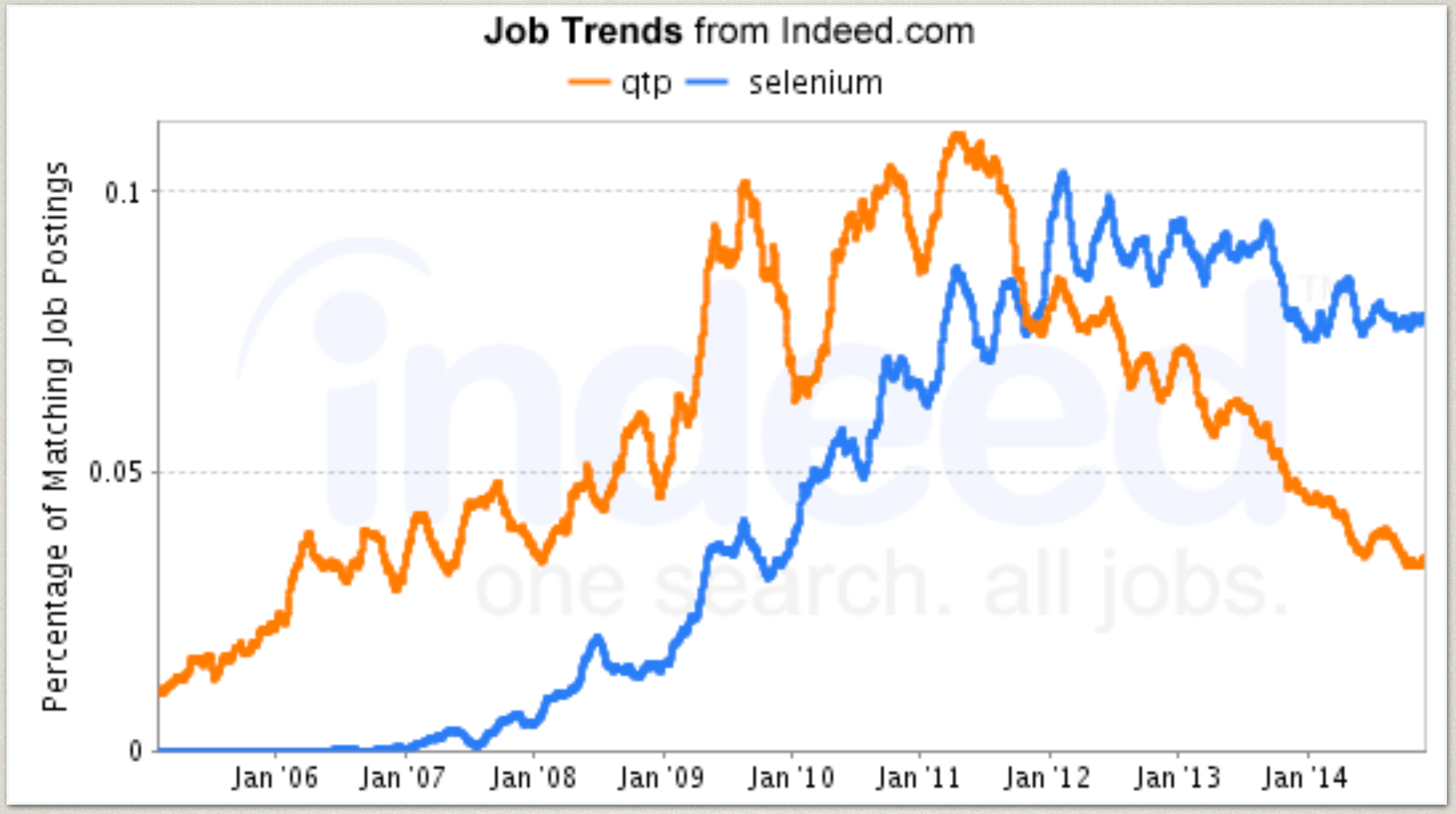
```
username = driver.find_element(By.ID, "foo")
username.send_keys("camembert")
```

SPI: Procedural and stateless

```
find_element(using="id", value="username")
send_keys(element="e0", value="camembert")
```

WEBDRIVER INTERNALS




Job Trends from Indeed.com
qtp
selenium
Percentage of Matching Job Postings
0.1
0.05
0
Jan '06
Jan '07
Jan '08
Jan '09
Jan '10
Jan '11
Jan '12
Jan '13
Jan '14
indeed
one search. all jobs.

DRIVERS

	Third Party	Selenium
Chrome	✓	
Firefox		✓
Internet Explorer		✓
Safari		✓
Opera (Presto)	✓	
Opera (Chrome)	✓	

DRIVERS

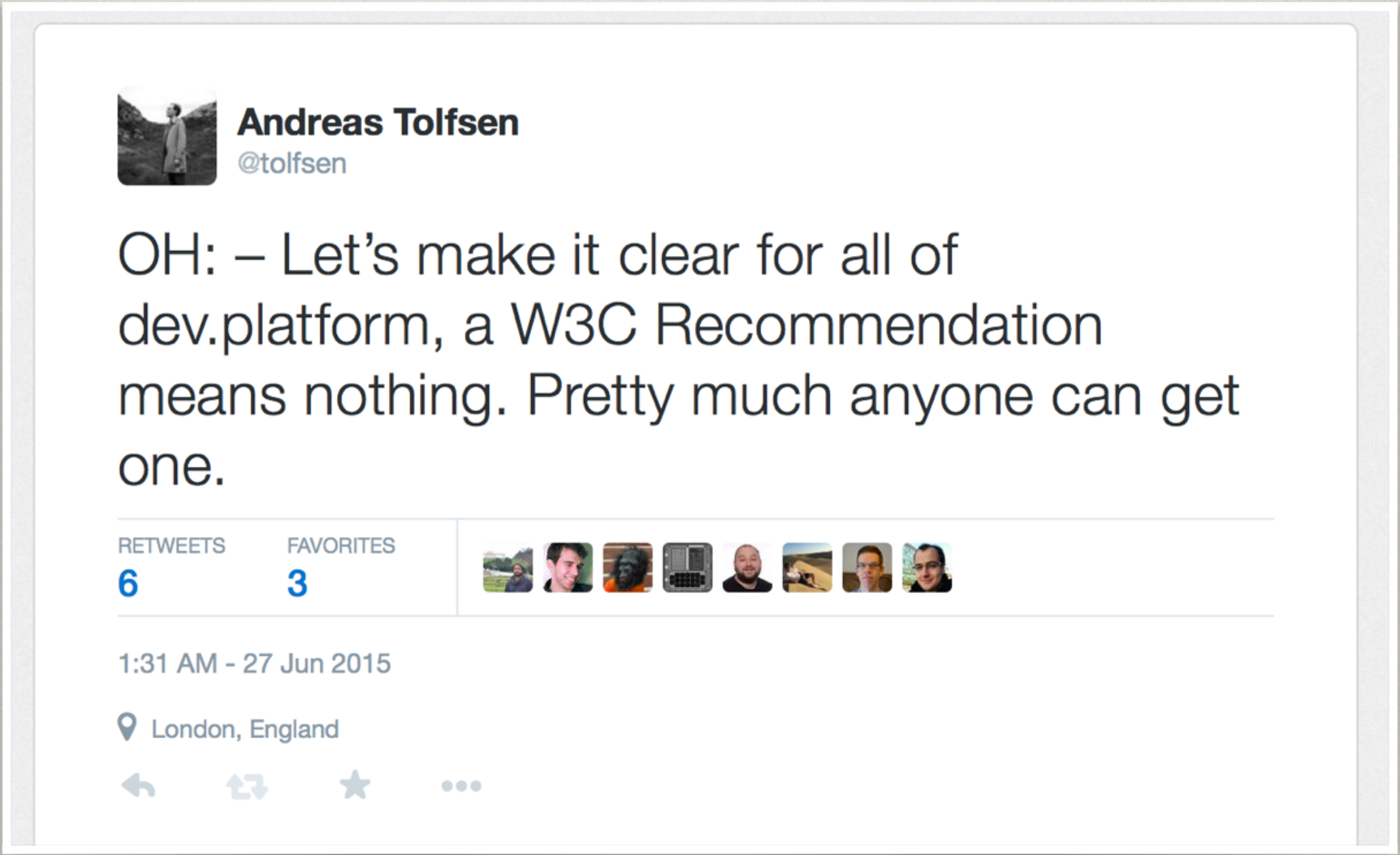
	Third Party	Selenium
Chrome	✓	
Firefox	←	✓
Internet Explorer	←	✓
Safari		✓
Opera (Presto)	✓	
Opera (Chrome)	✓	

DRIVERS

	Third Party	Selenium
Chrome	✓	
Firefox	✓ ←	
Internet Explorer	✓ ←	
Safari		✓
Opera (Presto)	✓	
Opera (Chrome)	✓	

WHY A SPEC?

- Most W3C specifications are not worth the paper they're written on
- Best metric of success is adoption
- Vendors want interop
- Support in Selenium for some browsers is an uphill battle



Andreas Tolfsen
@tolfsen

OH: – Let’s make it clear for all of dev.platform, a W3C Recommendation means nothing. Pretty much anyone can get one.

RETWEETS
6

FAVORITES
3

1:31 AM - 27 Jun 2015

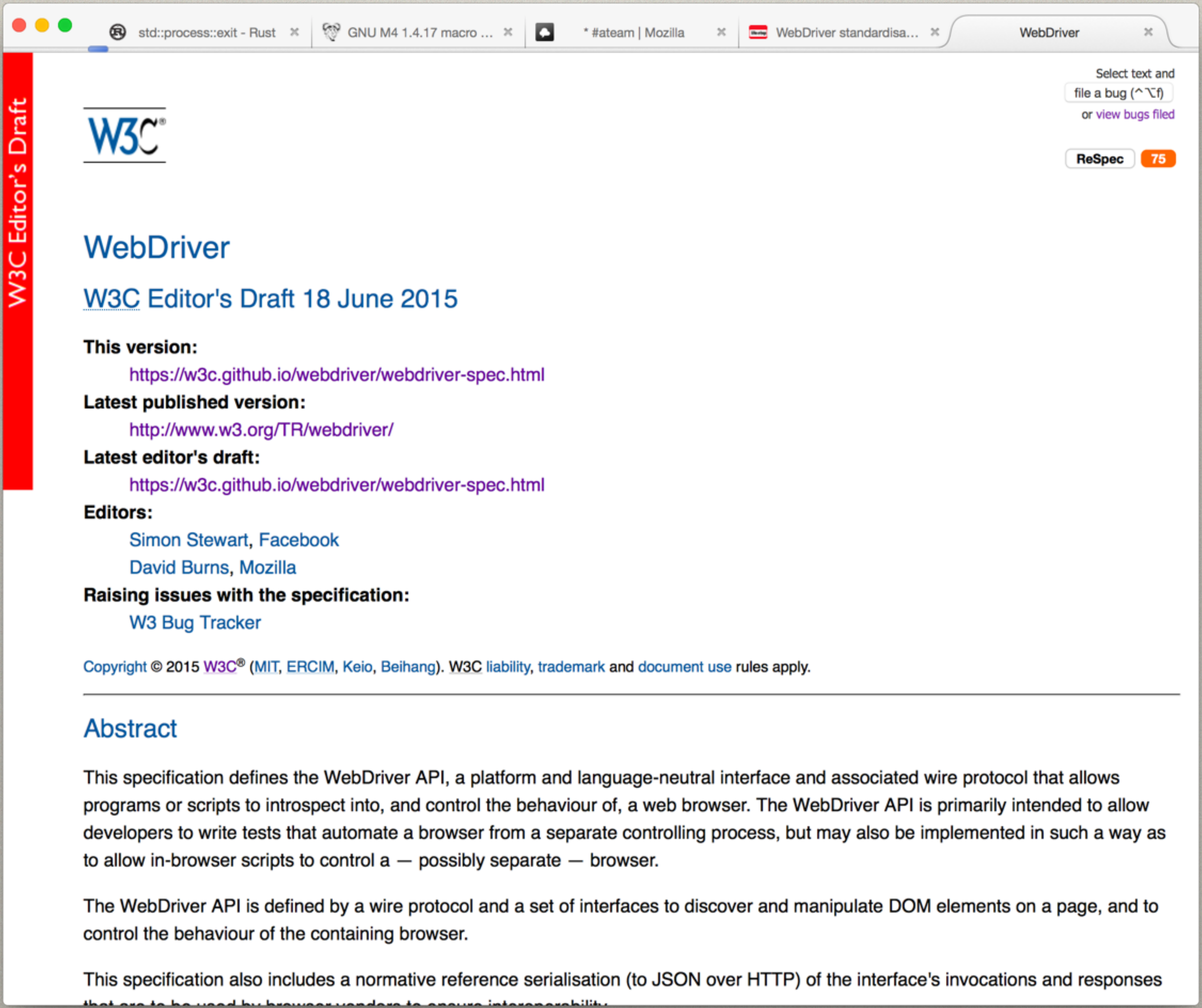
London, England

WEB PLATFORM CONCERNS

- From interlinked hypertext documents to an advanced application delivery platform
- Imposes new requirements on underlying technologies
- Robust and interoperable
- Interoperability only guaranteed through intra-browser testing
- WebDriver happens to be centrepiece of this effort

WEBDRIVER SPEC

- WebDriver is being standardised in the W3C
- Turning WebDriver from being
  the *de facto* solution for browser automation
  into the *du jour* solution
- Vendors involved: Google, Microsoft, Mozilla
    - Plus a number of invited experts
- WG has been meeting since October 2012
  (Lyon, France) and has had six meetings



std::process::exit - Rust
GNU M4 1.4.17 macro ...
* #ateam | Mozilla
WebDriver standardisa...
WebDriver

W3C Editor's Draft

Select text and
file a bug (^⌥f)
or view bugs filed

ReSpec 75

WebDriver

W3C Editor's Draft 18 June 2015

This version:
https://w3c.github.io/webdriver/webdriver-spec.html
Latest published version:
http://www.w3.org/TR/webdriver/
Latest editor's draft:
https://w3c.github.io/webdriver/webdriver-spec.html
Editors:
Simon Stewart, Facebook
David Burns, Mozilla
Raising issues with the specification:
W3 Bug Tracker

Abstract

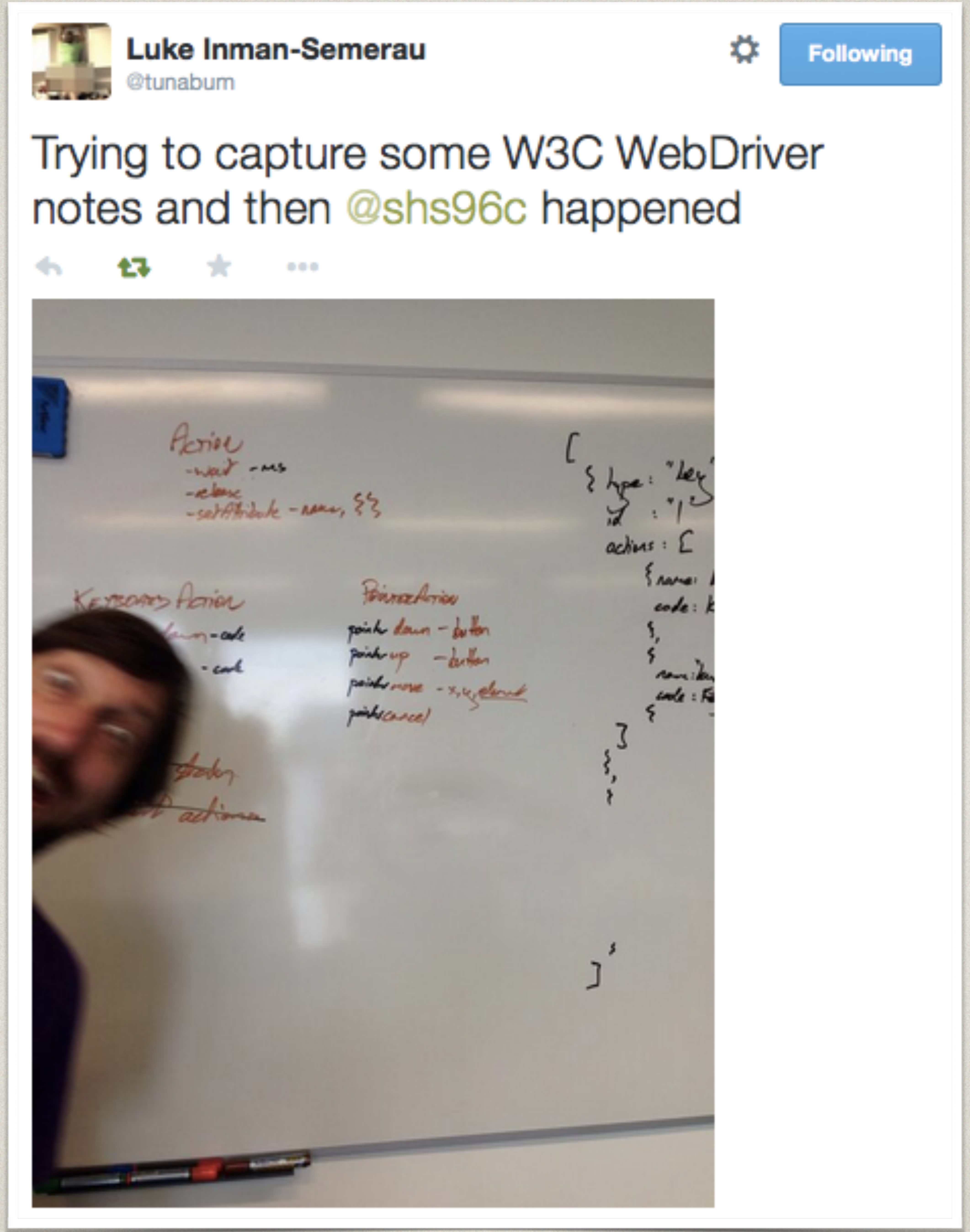
This specification defines the WebDriver API, a platform and language-neutral interface and associated wire protocol that allows programs or scripts to introspect into, and control the behaviour of, a web browser. The WebDriver API is primarily intended to allow developers to write tests that automate a browser from a separate controlling process, but may also be implemented in such a way as to allow in-browser scripts to control a — possibly separate — browser.

The WebDriver API is defined by a wire protocol and a set of interfaces to discover and manipulate DOM elements on a page, and to control the behaviour of the containing browser.

This specification also includes a normative reference serialisation (to JSON over HTTP) of the interface's invocations and responses

W3C Browser Tools- and Testing WG in Santa Clara, California

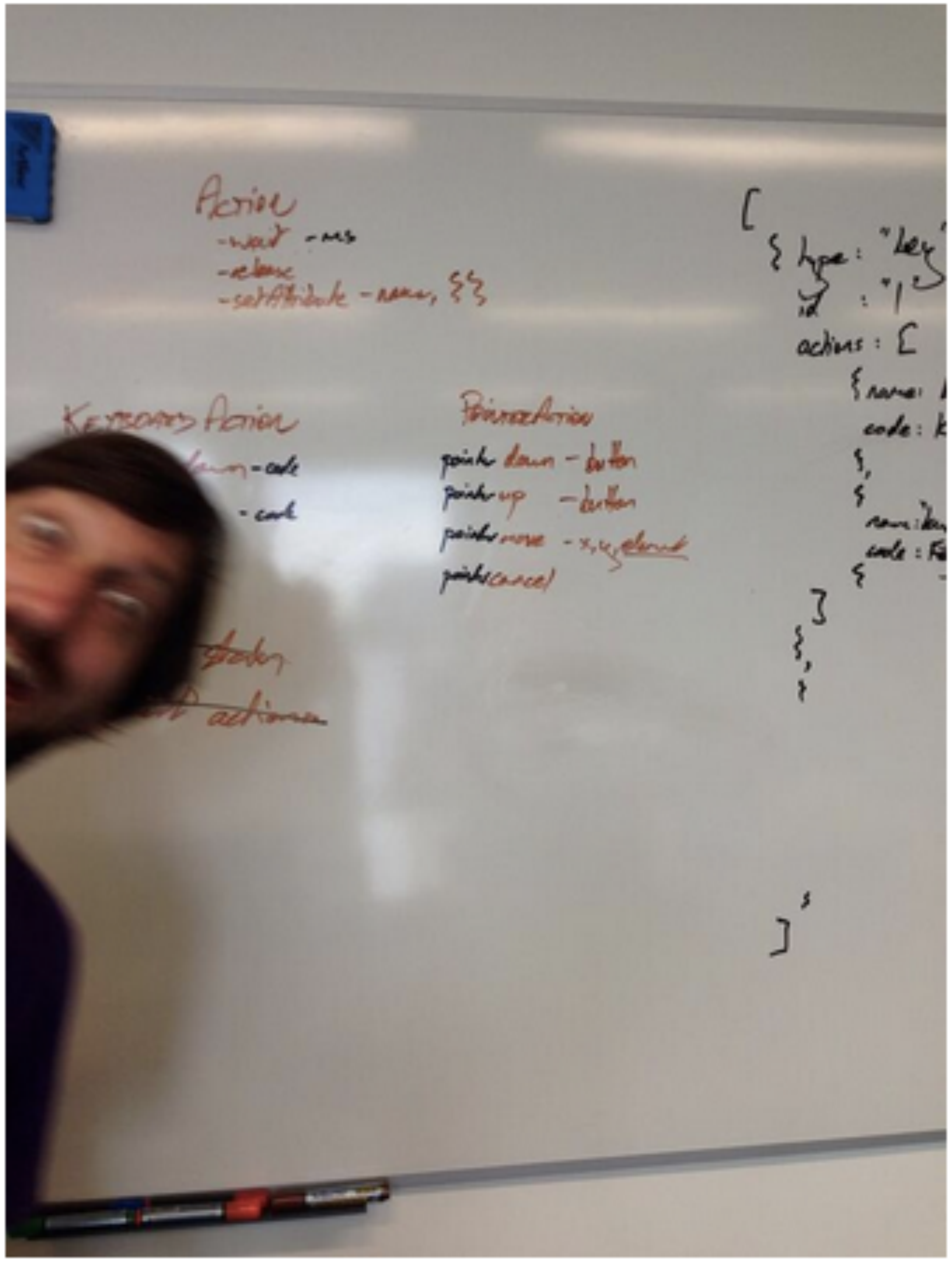
mozilla



Luke Inman-Semerau
@tunabum

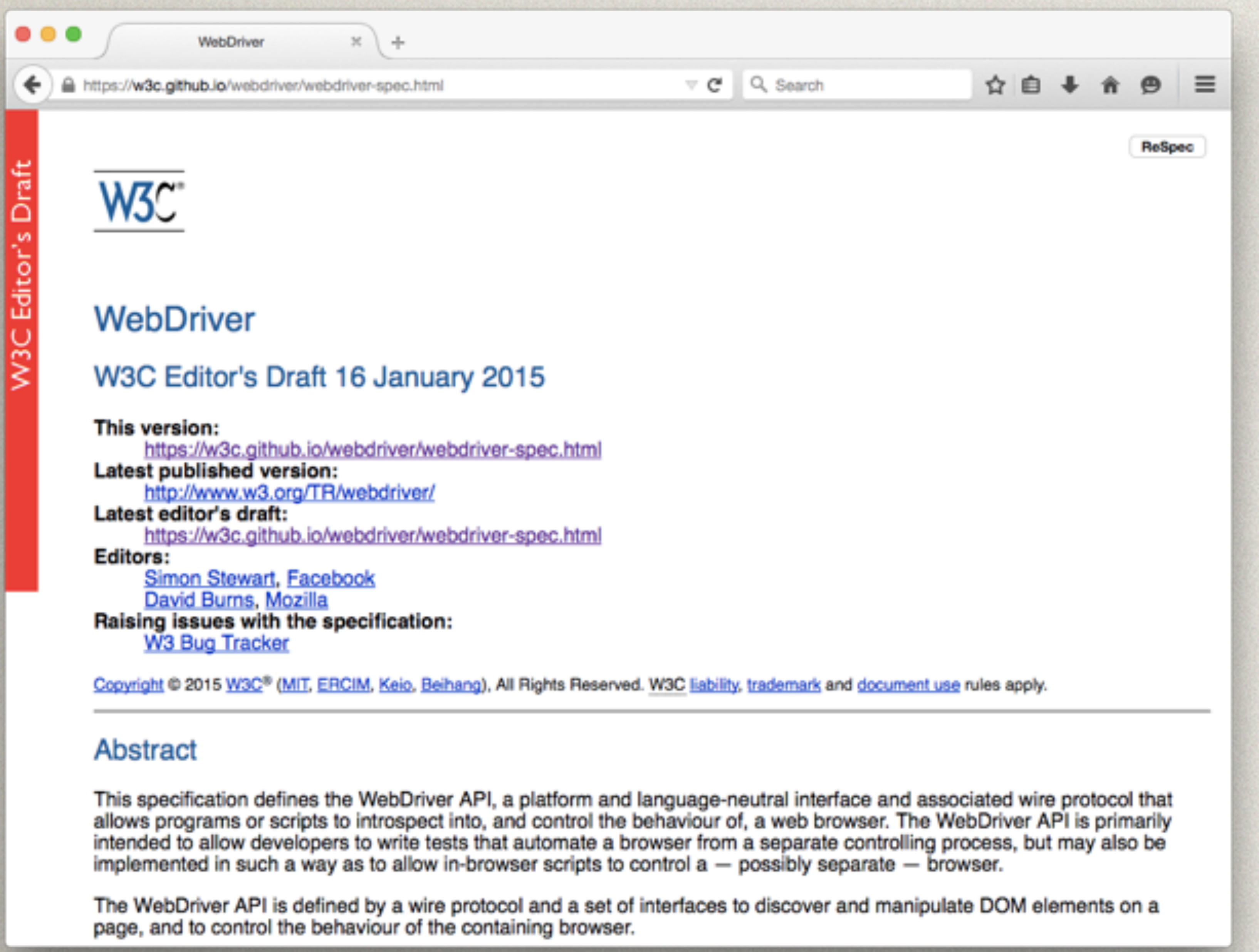
Following

Trying to capture some W3C WebDriver notes and then @shs96c happened



mozilla

W3C WEBDRIVER SPEC



WebDriver

W3C Editor's Draft 16 January 2015

This version:
https://w3c.github.io/webdriver/webdriver-spec.html
Latest published version:
http://www.w3.org/TR/webdriver/
Latest editor's draft:
https://w3c.github.io/webdriver/webdriver-spec.html
Editors:
Simon Stewart, Facebook
David Burns, Mozilla
Raising issues with the specification:
W3 Bug Tracker

Copyright © 2015 W3C® (MIT, ERCIM, Keio, Beihang), All Rights Reserved. W3C liability, trademark and document use rules apply.

Abstract

This specification defines the WebDriver API, a platform and language-neutral interface and associated wire protocol that allows programs or scripts to introspect into, and control the behaviour of, a web browser. The WebDriver API is primarily intended to allow developers to write tests that automate a browser from a separate controlling process, but may also be implemented in such a way as to allow in-browser scripts to control a — possibly separate — browser.

The WebDriver API is defined by a wire protocol and a set of interfaces to discover and manipulate DOM elements on a page, and to control the behaviour of the containing browser.

- Mostly about defining WebDriver as-is
- Possibility to refine parts of the **wire protocol**
- Tighten up language around behaviour
- New features in level 2 or “extension” modules

CHANGES (1/2)

- Wire protocol (SPI) changes
  - Will make existing 2.x client bindings incompatible!
  - API will remain mostly unchanged
- Actions, multi-actions, parallel actions, touch
- Errors will be more consistent
- Full page screenshots will go away

CHANGES (2/2)

- *getElementRect*
- `<input type=file multiple>`
- *switchToWindow* will only take handle as argument
- Stop conflating attributes and properties in *getElementAttribute*

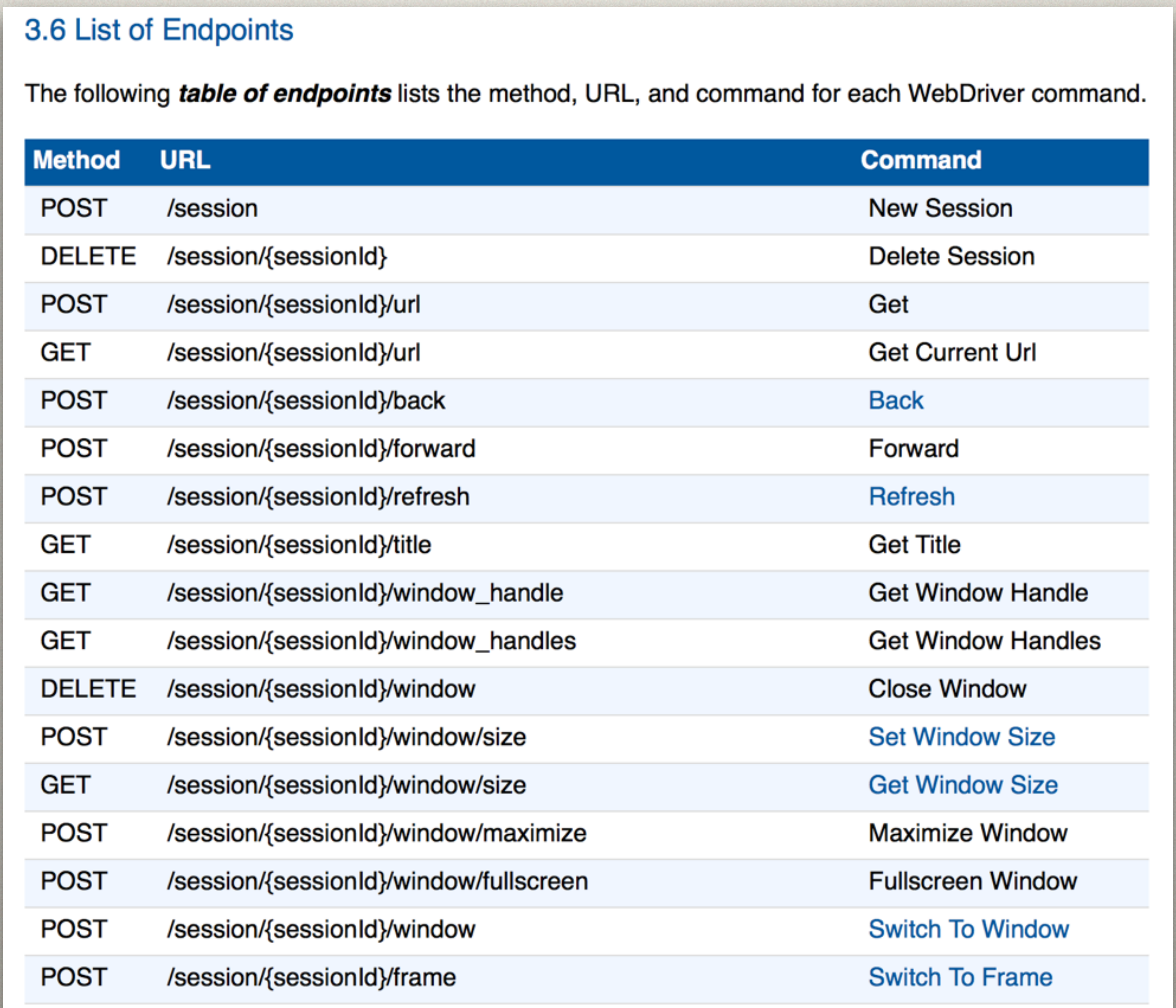
TERMINOLOGY

Local End

This represents the client side of the protocol, which is usually in the form of language-specific libraries providing an API on top of the WebDriver protocol. This specification does not place any restrictions on the details of those libraries above the level of the wire protocol.

mozilla

DIGRESSION: WIRE PROTOCOL



3.6 List of Endpoints

The following ***table of endpoints*** lists the method, URL, and command for each WebDriver command.

Method	URL	Command
POST	/session	New Session
DELETE	/session/{sessionId}	Delete Session
POST	/session/{sessionId}/url	Get
GET	/session/{sessionId}/url	Get Current Url
POST	/session/{sessionId}/back	Back
POST	/session/{sessionId}/forward	Forward
POST	/session/{sessionId}/refresh	Refresh
GET	/session/{sessionId}/title	Get Title
GET	/session/{sessionId}/window_handle	Get Window Handle
GET	/session/{sessionId}/window_handles	Get Window Handles
DELETE	/session/{sessionId}/window	Close Window
POST	/session/{sessionId}/window/size	Set Window Size
GET	/session/{sessionId}/window/size	Get Window Size
POST	/session/{sessionId}/window/maximize	Maximize Window
POST	/session/{sessionId}/window/fullscreen	Fullscreen Window
POST	/session/{sessionId}/window	Switch To Window
POST	/session/{sessionId}/frame	Switch To Frame

TERMINOLOGY

Remote End

The remote end hosts the server side of the protocol. Defining the behaviour of a remote end in response to the WebDriver protocol forms the largest part of this specification.

TERMINOLOGY

Intermediary Node

Intermediary nodes are those that act as proxies, implementing both the client and server sides of the protocol. Intermediary nodes must be black-box indistinguishable from a *remote end* from the point of view of *local end* and so are bound by the requirements on a *remote end* in terms of the wire protocol. However they are not expected to implement commands directly.

TERMINOLOGY

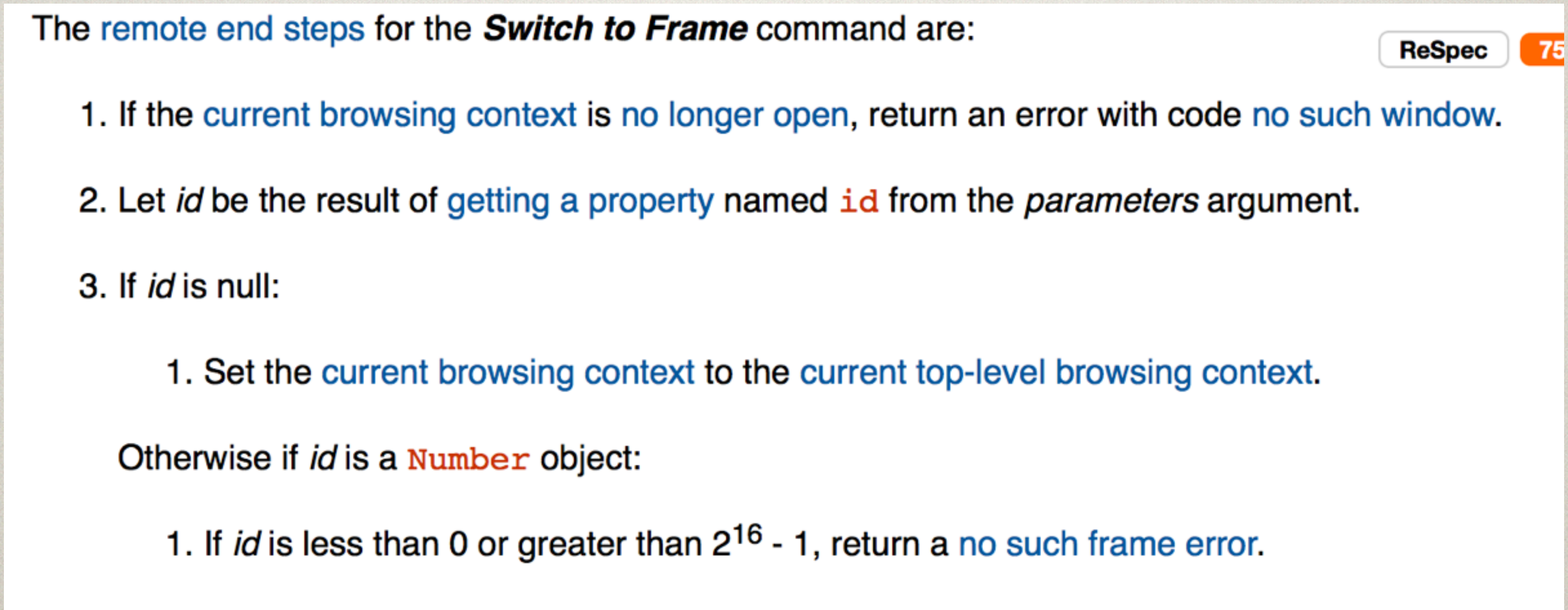
Endpoint Node

An endpoint node is the final *remote end* in a chain of nodes that is not an *intermediary node*. The endpoint node is implemented by a user agent or a similar program. An endpoint node must be, like *intermediary nodes*, indistinguishable from a *remote end*.

mozilla

ALGORITHMS

Spec written using step-by-step instructions, or algorithms that carry **no normative significance**, as long as implementations produce equivalent output to what is described.



The remote end steps for the ***Switch to Frame*** command are:

ReSpec 75

1. If the current browsing context is no longer open, return an error with code no such window.
2. Let *id* be the result of getting a property named `id` from the *parameters* argument.
3. If *id* is null:
    1. Set the current browsing context to the current top-level browsing context.

   Otherwise if *id* is a `Number` object:

    1. If *id* is less than 0 or greater than 2^{16} - 1, return a no such frame error.

    1. Set the current browsing context to the current top-level browsing context.

   Otherwise if *id* is a `Number` object:

    1. If *id* is less than 0 or greater than 2^{16} - 1, return a no such frame error.

    2. Let *window* be the associated window of the current browsing context's active document.

    3. If *id* is not a supported property index of *window*, return no such frame error.

    4. Let *child window* be the `WindowProxy` object obtained by determining the value of an indexed property of *window* with index *id*.

    5. Set the current browsing context to *new window*'s browsing context.

   Otherwise, if *id* represents a web element:

    1. Let *element* be the element represented by *id*.

    2. If *element* is not a `frame` or `iframe` element, return a no such frame error.

    3. Set the current browsing context to element's nested browsing context.

   Otherwise:

    1. Return no such frame error.

4. Return success with data null.

PROCESSING MODEL

- Drivers mandated to implement HTTP-over-TCP, serving a defined list of **endpoints**
- The **remote end** runs a main loop that handles incoming requests*
- Matches the request by its **method** and **URL**
- Special cases for **New Session** and **POST's**, defines **error handling**
- Runs the **remote end steps** for the command
- Serialises and **writes data** back on the connection

After such a connection has been established, a remote end MUST run the following steps:

1. Read bytes from the connection until a complete HTTP request can be constructed from the data. Let *request* be a request object constructed from the received data, according to the requirements of [RFC7230].

2. Let *request match* be the result of the algorithm to match a request with *request*'s method and url as arguments.

3. If *request match* is of type error, send an error with *request match*'s error code and jump to step 1.

   Otherwise, let *command*, *session id* and *element id* be *request match*'s data.

4. If *command* is not New Session:

   1. If *session id* is not equal to the id of any session in the list of active sessions, send an error with error code invalid session id, then jump to step 1 in this overall algorithm.

      Otherwise, let the current session be the session with id *session id*.

5. If *request*'s method is POST:

   1. Let *parse result* be the result of parsing as JSON with *request*'s body as the argument.

   2. If *parse result* is an error or if it is a success but its associated data is not an Object object, send an error with error code invalid argument and jump back to step 1 in this overall algorithm.

      Otherwise, let *parameters* be*parse result*'s data.

   Otherwise, let *parameters* be null.

6. Let *response data* be a command response object obtained by running the remote end steps for *command* with arguments *element id* and *parameters*.

7. If *response data* is an error, send an error with error code equal to *response data*'s error code.

   Otherwise send a response with status 200 and *response data*'s data.

8. Jump to step 1.

mozilla

COMPATIBILITY

- Does not specify transport mechanism a driver should use
- But, since October 2014 it **does** mandate implementors to ship an HTTPD
  - Most vendors will ship a shim that implements the W3C WebDriver HTTP/JSON-over-TCP wire protocol
  - Mozilla’s is written in Rust

mozilla

PIPELINING

Commands written with such care that they can potentially be pipelined into a single endpoint.

Why? **Latency**, **uniformity**, and **future-proofing**.

```
[
  {name: "Click Element", {ELEMENT: …}},
  {name: "Pointer Move", {x: …, y: …}},
  {name: "Execute Script", {script: …}},
  {name: "Pointer Down", {}},
  {name: "Pointer Up", {}}
]
```

CONFORMANCE TESTING

- Web Platform Tests (WPT)
- Tests for the open web platform
- Collated by W3C
- Contributions from individuals and browser vendors
- http://testthewebforward.org/docs
- https://github.com/w3c/web-platform-tests

mozilla

WEB PLATFORM TESTS (1/2)

- Tests browsers for compatibility against standards
  (HTML, DOM, ECMAScript/JavaScript, URL, XHR, CORS, CSS, Encoding, …)
- Specs require tests to go become a **recommendation**
- Ethos: Interoperability through testing
- Standardising WebDriver means exciting things for web standards

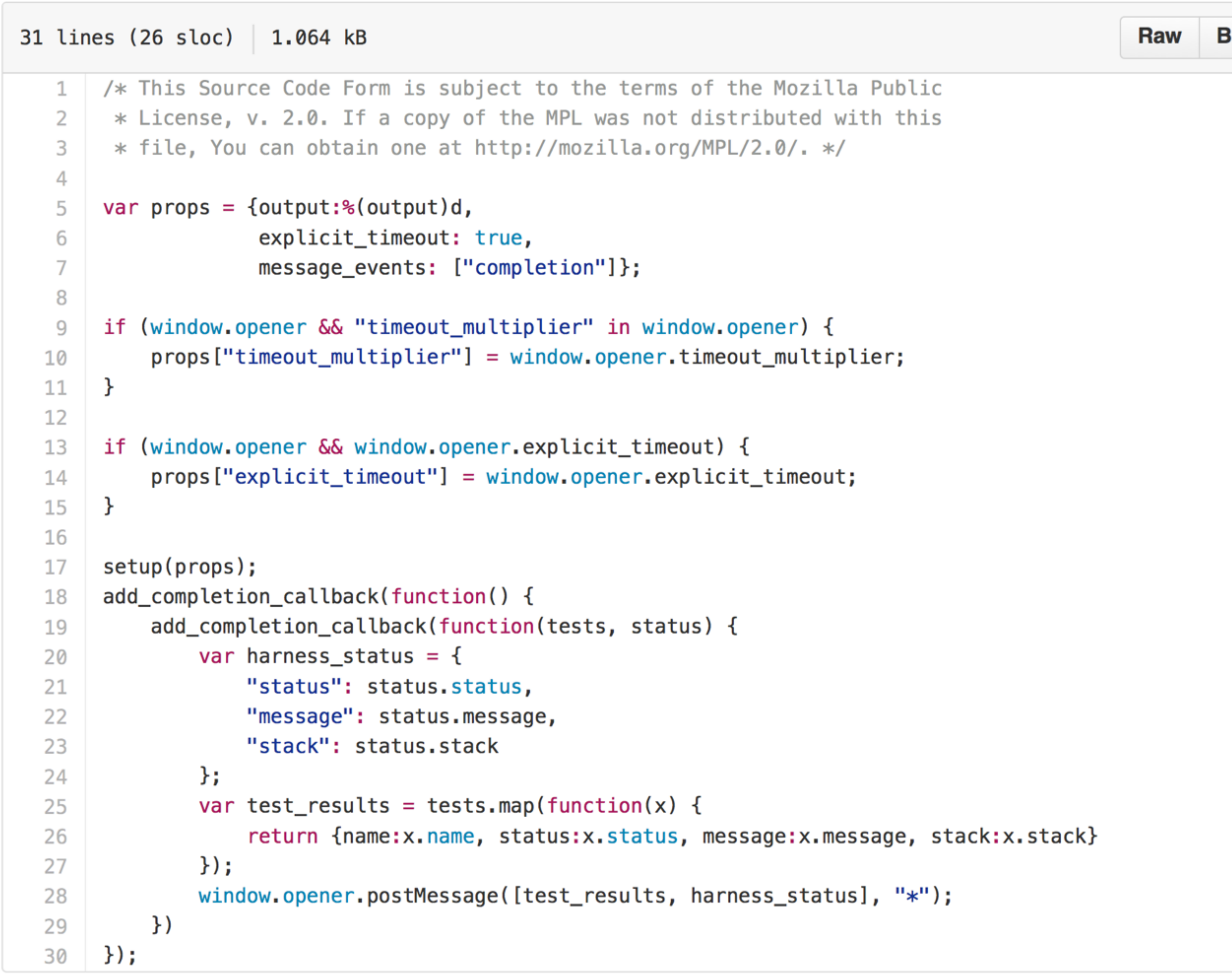
mozilla

WEB PLATFORM TESTS (2/2)

- *WPT* are vendor-neutral, automated tests run in continuous integration that anyone can contribute to
- *wptrunner* uses WebDriver to drive automation of these tests in IE, Chrome, Firefox, Firefox OS, and Servo
- WebDriver is the **missing link** in functional browser testing

TEST CASES

- Browser binary process control
- Profile management, environment guarantees
- *testharness.js*
    - Loads harness, which triggers a child window where the tests are run, posting messages back to the parent window (postMess



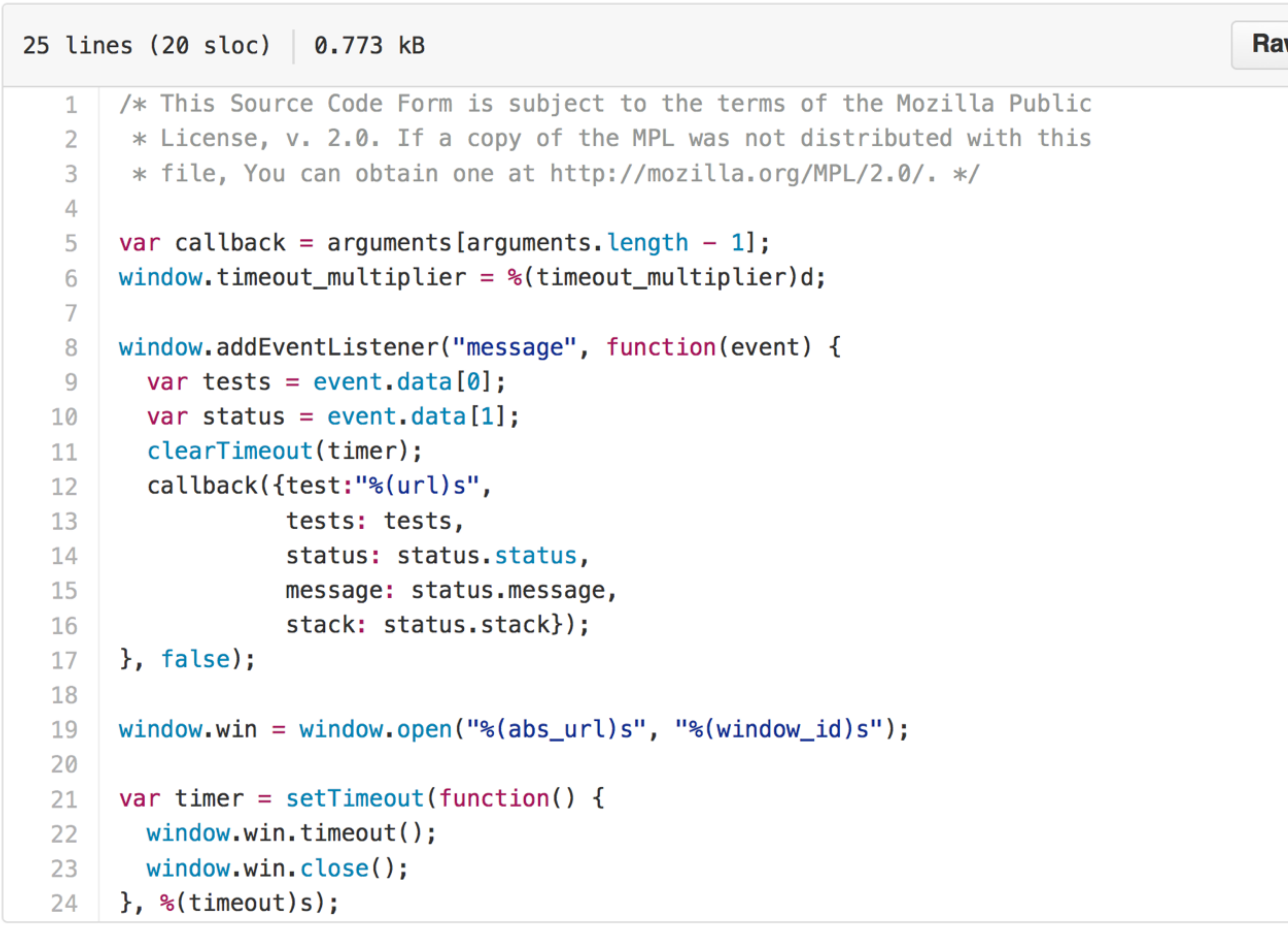
```
/* This Source Code Form is subject to the terms of the Mozilla Public
 * License, v. 2.0. If a copy of the MPL was not distributed with this
 * file, You can obtain one at http://mozilla.org/MPL/2.0/. */

var props = {output:%(output)d,
             explicit_timeout: true,
             message_events: ["completion"]};

if (window.opener && "timeout_multiplier" in window.opener) {
    props["timeout_multiplier"] = window.opener.timeout_multiplier;
}

if (window.opener && window.opener.explicit_timeout) {
    props["explicit_timeout"] = window.opener.explicit_timeout;
}

setup(props);
add_completion_callback(function() {
    add_completion_callback(function(tests, status) {
        var harness_status = {
            "status": status.status,
            "message": status.message,
            "stack": status.stack
        };
        var test_results = tests.map(function(x) {
            return {name:x.name, status:x.status, message:x.message, stack:x.stack}
        });
        window.opener.postMessage([test_results, harness_status], "*");
    })
});
```



25 lines (20 sloc) | 0.773 kB

```
/* This Source Code Form is subject to the terms of the Mozilla Public
 * License, v. 2.0. If a copy of the MPL was not distributed with this
 * file, You can obtain one at http://mozilla.org/MPL/2.0/. */

var callback = arguments[arguments.length - 1];
window.timeout_multiplier = %(timeout_multiplier)d;

window.addEventListener("message", function(event) {
  var tests = event.data[0];
  var status = event.data[1];
  clearTimeout(timer);
  callback({test:"%(url)s",
            tests: tests,
            status: status.status,
            message: status.message,
            stack: status.stack});
}, false);

window.win = window.open("%(abs_url)s", "%(window_id)s");

var timer = setTimeout(function() {
  window.win.timeout();
  window.win.close();
}, %(timeout)s);
```

WHAT DOES THIS MEAN FOR SELENIUM?

- Future of the project centres around the W3C specification
- No more drivers maintained by Selenium
- Selenium WebDriver protocol will be decommissioned
  - Burning question: How to handle backwards compatibility?

SELENIUM 3

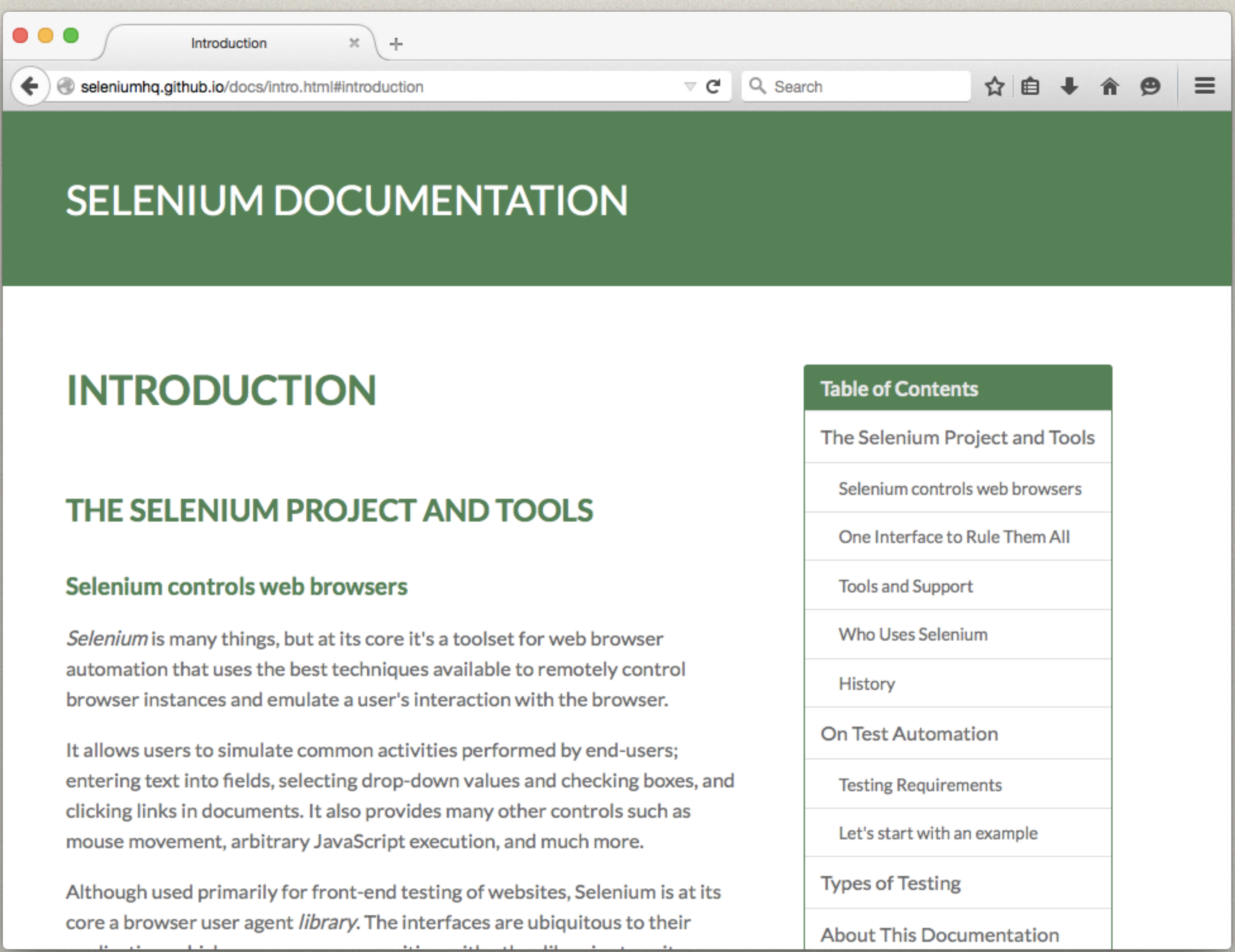
- Selenium RC will be deprecated
  - Will be split out as separate download
  - Absolutely no active feature development
  - Recommendation: Use *SeleniumBackedWebDriver*
- Uncertain future for Selenium IDE
- Cleanups in API?
- New documentation

SELENIUM 3 (CONTD.)

- Cleanups in API?
  - Using WebDriver after quit should yield error
  - Actions to have a single endpoint
  - Required capabilities in all API bindings
  - Clean up constructors for FirefoxDriver/ChromeDriver classes
  - Delete unnecessary cruft
  - Cleaner end-point for RC emulation

SELENIUM 4

- Spec compliance for language bindings
  - What to do about Selenium 2 bindings connecting to a version 4 server?
  - Shim for backwards compatibility?
- Selenium will become **language bindings** and **test suite**
- Drivers the responsibility of browser vendors, with Selenium providing the ecosystem



SELENIUM DOCUMENTATION

INTRODUCTION

THE SELENIUM PROJECT AND TOOLS

Selenium controls web browsers

Selenium is many things, but at its core it's a toolset for web browser automation that uses the best techniques available to remotely control browser instances and emulate a user's interaction with the browser.

It allows users to simulate common activities performed by end-users; entering text into fields, selecting drop-down values and checking boxes, and clicking links in documents. It also provides many other controls such as mouse movement, arbitrary JavaScript execution, and much more.

Although used primarily for front-end testing of websites, Selenium is at its core a browser user agent *library*. The interfaces are ubiquitous to their

Table of Contents

https://seleniumhq.github.io/docs

NOT IN SPEC

- Mobile
  - Level 1 is **foundations**, further improvements could come as module extensions
- Web API?
  - Battery, telephony, SMS, geolocation, screen orientation, push, permissions, payment
- Logging
- Performance

WEBDRIVER DRAWBACKS

- Synchronous, blocking API
- Limited to one connection
- Data intensive, no opt-in to just what you care about
- Highly specific to end-user emulation
  and browser automation
- Lack of generality
- Expensive protocol and transport mechanism
- Many of its primitives make no sense

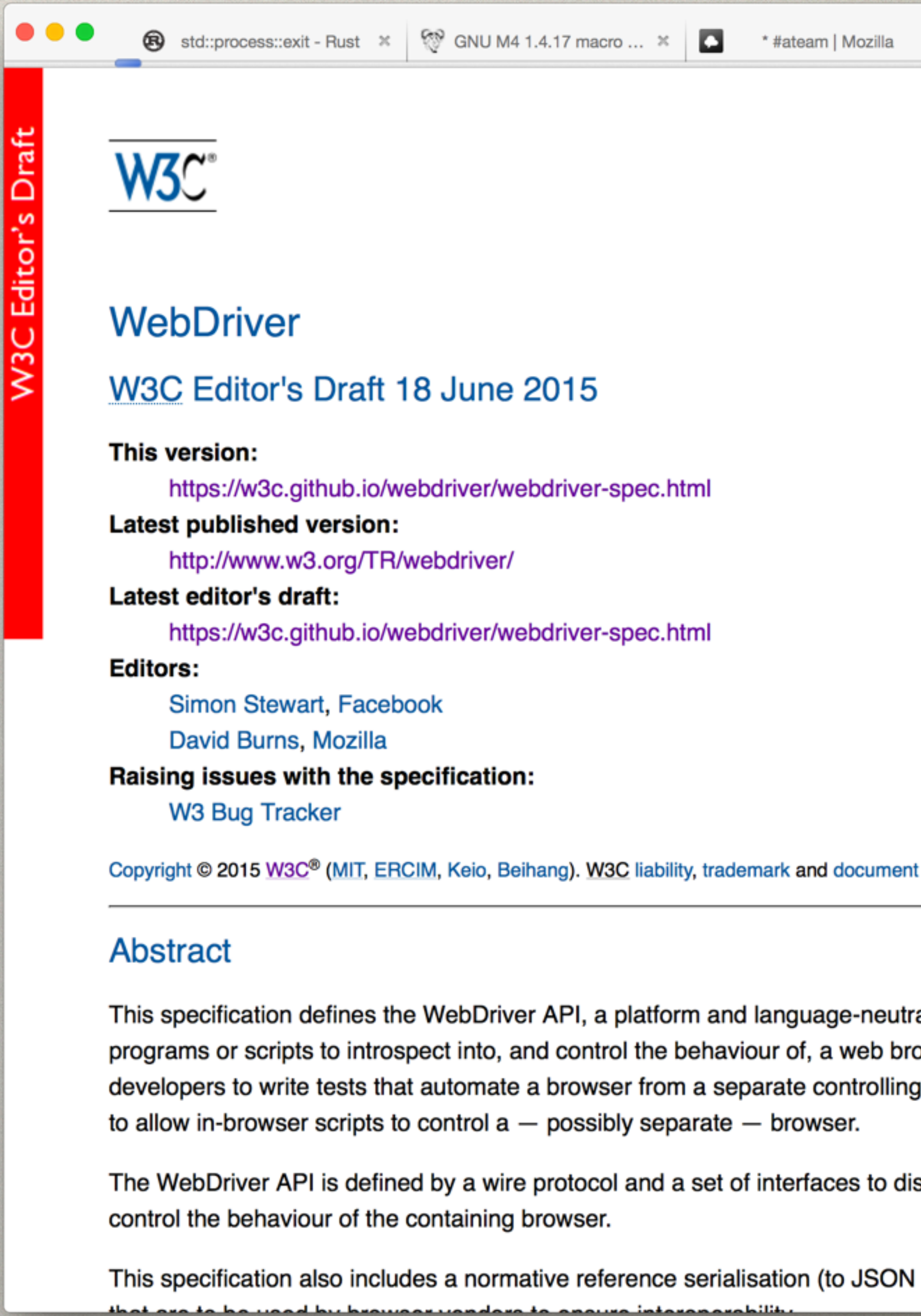
ROADMAP

- Working Group meeting again at the *W3C Technical Plenary/Advisory Committee Meetings Week* (TPAC) in Sapporo, Japan in October
- Goal is to finish most of it before then,
  so meetings can be about the finer adjustments
- Work is already well under way – chances we may have something quite solid by end of year
- Test suite + push to recommendation Q1 2016

mozilla

HELPING OUT

- We could use help and input
- Bugs can be filed
- Tests
- Implementations



WebDriver

W3C Editor's Draft 18 June 2015

This version:
https://w3c.github.io/webdriver/webdriver-spec.html

Latest published version:
http://www.w3.org/TR/webdriver/

Latest editor's draft:
https://w3c.github.io/webdriver/webdriver-spec.html

Editors:
Simon Stewart, Facebook
David Burns, Mozilla

Raising issues with the specification:
W3 Bug Tracker

Copyright © 2015 W3C® (MIT, ERCIM, Keio, Beihang). W3C liability, trademark and document

Abstract

This specification defines the WebDriver API, a platform and language-neutra
programs or scripts to introspect into, and control the behaviour of, a web bro
developers to write tests that automate a browser from a separate controlling
to allow in-browser scripts to control a — possibly separate — browser.

The WebDriver API is defined by a wire protocol and a set of interfaces to dis
control the behaviour of the containing browser.

This specification also includes a normative reference serialisation (to JSON

mozilla

THANKS!

Andreas Tolfsen

@tolfsen