

Common Lisp Webdriver Client

A.U. Thor <bug-sample@gnu.org>

Table of Contents

1	Introduction	1
2	Usage	2
2.1	Sessions	2
2.2	Capabilities	3
2.2.1	alwaysMatch	4
2.2.2	firstMatch	4
2.2.3	Combining alwaysMatch and firstMatch	4
2.2.4	List of capabilities	5
2.2.5	Vendor-specific capabilities	5
2.3	Actions	5
3	Installation	7
4	Utils	8
4.1	Interactive session	8
4.2	Utils API conventions	8
4.3	Waiting for the reaction	9
4.4	Running tests	9
5	API	10
5.1	WEBDRIVER-CLIENT package	10
5.2	WEBDRIVER-CLIENT-UTILS package	19
6	Index	21

1 Introduction

CL Webdriver Client is client library for WebDriver.

WebDriver is a remote control interface that enables introspection and control of user agents. It provides a platform- and language-neutral wire protocol as a way for out-of-process programs to remotely instruct the behavior of web browsers.

Provided is a set of interfaces to discover and manipulate DOM elements in web documents and to control the behavior of a user agent. It is primarily intended to allow web authors to write tests that automate a user agent from a separate controlling process, but may also be used in such a way as to allow in-browser scripts to control a — possibly separate — browser.

See W3C Webdriver spec (<https://www.w3.org/TR/webdriver>).

2 Usage

```
;; see examples/*.lisp and t/*.lisp
(in-package :cl-user)

(eval-when (:compile-toplevel :load-toplevel :execute)
  (ql:quickload :cl-webdriver-client))

(defpackage go-test
  (:use :cl :webdriver-client))

(in-package :go-test)

(defparameter *code* "
package main
import \"fmt\"

func main() {
    fmt.Print(\"Hello WebDriver!\")
}")

(with-session ()
  (setf (url) "http://play.golang.org/?simple=1")
  (let ((elem (find-element "#code" :by :css-selector)))
    (element-clear elem)
    (element-send-keys elem *code*))
  (let ((btn (find-element "#run")))
    (element-click btn))

  (loop
    with div = (find-element "#output")
    for ouput = (element-text div)
    while (equal ouput "Waiting for remote server...")
    do (sleep 0.1)
    finally (print ouput)))
```

2.1 Sessions

A session is equivalent to a single instantiation of a particular user agent, including all its child browsers. WebDriver gives each session a unique session ID that can be used to differentiate one session from another, allowing multiple user agents to be controlled from a single HTTP server, and allowing sessions to be routed via a multiplexer (known as an intermediary node).

A WebDriver session represents the connection between a local end and a specific remote end.

A session is started when a New Session is invoked. It is an error to send any commands before starting a session, or to continue to send commands after the session has been closed.

Maintaining session continuity between commands to the remote end requires passing a session ID.

A session is torn down at some later point; either explicitly by invoking Delete Session, or implicitly when Close Window is called at the last remaining top-level browsing context.

An intermediary node will maintain an associated session for each active session. This is the session on the upstream neighbor that is created when the intermediary node executes the New Session command. Closing a session on an intermediary node will also close the session of the associated session.

All commands, except New Session and Status, have an associated current session, which is the session in which that command will run.

2.2 Capabilities

WebDriver capabilities are used to communicate the features supported by a session. A client may also use capabilities to define which features it requires the driver to satisfy when creating a new session.

When a WebDriver session is created it returns a set of capabilities describing the negotiated, effective capabilities of the session. Some of the capabilities included in this set are standard and shared between all browsers, but the set may also contain browser-specific capabilities and these are always prefixed. Capabilities negotiation

Capabilities can be used to require a driver that supports a certain subset of features. This can be used to require certain browser features, such as the ability to resize the window dimensions, but is also used in distributed environments to select a particular browser configuration from a matrix of choices.

Selecting a particular web browser or platform only makes sense when you use a remote WebDriver. In this case the client makes contact with WebDriver through one or more intermediary nodes which negotiates which driver to return to you based on the capabilities it receives.

The capabilities object is a selection mechanism that limits which driver configurations the server will return. If you request a Firefox instance using `browserName` and Firefox is not installed on the remote, or macOS from a remote that only supports Linux, you may be out of luck. But occasionally you may not care which specific operating system or web browser your session has: you just want a session that has some capability.

The selection process, or the capabilities negotiation, is done through `alwaysMatch` and `firstMatch`.

In *CL-WEBDRIVER-CLIENT* capabilities are created using [MAKE-CAPABILITIES], page 11. The key argument `:always-match` should be the 'alwaysMatch' capabilities as an association list. The key argument `:first-match` are the 'firstMatch' capabilities, a list of association lists.

Also, capabilities are constructed from arguments in [MAKE-SESSION], page 12 and [WITH-SESSION], page 12.

2.2.1 alwaysMatch

As the name suggests, capabilities described inside the `alwaysMatch` capabilities object are features you require the session to have. If the server can not provide what features you require, it will fail.

If for example you ask for Firefox version 62 on a system that only has 60 installed, the session creation will fail:

```
(make-capabilities
  :always-match '(:browser-name . "firefox")
                (:browser-version . "60")))

or:

(with-session '(:always-match (:browser-name . "firefox")
                              (:browser-version . "60")))

... )
```

2.2.2 firstMatch

The `firstMatch` field accepts an array of capabilities objects which will be matched in turn until one matches what the server can provide, or it will fail.

This can be useful when you want a driver that runs on macOS or Linux, but not Windows:

```
(make-capabilities
  :first-match (list
    '(:platform-name . "macos"))
    '(:platform-name . "linux"))))
```

2.2.3 Combining alwaysMatch and firstMatch

`firstMatch` can of course be combined with `alwaysMatch` to narrow down the selection. If for example you a driver that runs on macOS or Linux but it has to be Firefox:

```
(make-capabilities
  :always-match '(:browser-name . "firefox")
  :first-match (list
    '(:platform-name . "macos"))
    '(:platform-name . "linux"))))
```

The previous example is exactly equivalent to putting the Firefox requirement in each `firstMatch` arm:

```
(make-capabilities
  :first-match (list
    '(:browser-name . "firefox") (:platform-name . "macos"))
    '(:browser-name . "firefox") (:platform-name . "linux"))))
```

Which you choose of the two preceding examples is not important, but it can matter when pass along browser configuration. To avoid unnecessarily repeating data, such as profiles, it is advisable to make use of `alwaysMatch` so that this data is only transmitted across the wire once:

```
(make-capabilities
  :always-match '(:browser-name . "firefox")
```

```

        ,(firefox-capabilities
          :profile "<base64 encoded profile>"
          :args #("--headless")
          :prefs '(("dom.ipc.processCount" . 8))
          :log '(:level . "trace"))))
      :first-match (list
        '(:platform-name . "macos"))
        '(:platform-name . "linux"))))

```

2.2.4 List of capabilities

- *:browser-name* – a string. The name of the browser to use. e.g. “chrome”, “firefox”.
- *:browser-version* – a string. The browser version required.
- *:platform-name* – a string. Identifies the operating system of the endpoint node. e.g. 'Linux', 'Windows'.
- *:accept-insecure-certs* – a boolean. The `acceptInsecureCerts` capability communicates whether expired or invalid TLS certificates are checked when navigating. If the capability is false, an insecure certificate error will be returned as navigation encounters domains with certificate problems. Otherwise, self-signed or otherwise invalid certificates will be implicitly trusted by the browser on navigation. The capability has effect for the lifetime of the session.
- *:page-load-strategy* – a string.
- *:proxy* – object. Defines the current session’s proxy configuration (<https://www.w3.org/TR/webdriver1/#dfn-proxy-configuration>).
- *:set-window-rect* – boolean. Indicates whether the remote end supports all of the commands in Resizing and Positioning Windows (<https://www.w3.org/TR/webdriver1/#resizing-and-positioning-windows>).
- *:timeouts* – object. Describes the timeouts (<https://www.w3.org/TR/webdriver1/#dfn-timeouts>) imposed on certain session operations.
- *:unhandled-prompt-behaviour* – string. Describes the current session’s user prompt handler.

Reference: <https://www.w3.org/TR/webdriver1/#capabilities>.

2.2.5 Vendor-specific capabilities

In addition to the standard capabilities WebDriver allows third-parties to extend the set of capabilities to match their needs. Browser vendors and suppliers of drivers typically use extension capabilities to provide configuration to the browser, but they can also be used by intermediaries for arbitrary blobs of information.

You can use [CHROME-CAPABILITIES], page 15 and [FIREFOX-CAPABILITIES], page 15 for these.

2.3 Actions

The Actions API provides a low-level interface for providing virtualised device input to the web browser. Conceptually, the Actions commands divide time into a series of ticks.

The local end sends a series of actions which correspond to the change in state, if any, of each input device during each tick. For example, pressing a key is represented by an action sequence consisting of a single key input device and two ticks, the first containing a `keyDown` action, and the second a `keyUp` action, whereas a pinch-zoom input is represented by an action sequence consisting of three ticks and two pointer input devices of type `touch`, each performing a sequence of actions `pointerDown`, followed by `pointerMove`, and then `pointerUp`.

See <https://www.w3.org/TR/webdriver/#actions> for the whole explanation.

To perform actions in *cl-webdriver-client* use [PERFORM-ACTIONS], page 13. That function implements a little language, with the following syntax:

Syntax:

```
actions ::= ({actions-input-source}*)
actions-input-source ::= (input-source-type {action}*)
input-source-type ::= :none | :pointer | :mouse | :pen | :touch | :key
action ::= pause | pointer-move | pointer-down | pointer-up | key-down | key-up
pause ::= (:pause duration)
pointer-move ::= (:pointer-move x y)
pointer-down ::= (:pointer-down button-number)
pointer-up ::= (:pointer-up button-number)
key-down ::= (:key-down key)
key-up ::= (:key-up key)
```

Arguments and values:

- *actions*—a list of actions-input-sources. One list for each type of input source that wants to be used.
- *actions-input-source*—a list. Specifies the list of actions to perform for a particular input source.
- *duration*—an integer. The time to pause in milliseconds.
- *key*—a string. A string with the character (e.g. “a”). Use [KEY], page 13 for entering special characters.
- *button-number*—an integer greater than or equal to 0.
- *x*—an integer. Horizontal screen coordinate.
- *y*—an integer. Vertical screen coordinate.

Examples:

```
(perform-actions '((:pen
  (:pointer-move 22 33)
  (:pause 2000)
  (:pointer-move 23 54))))
```

3 Installation

```
git clone https://github.com/copyleft/cl-webdriver-client ~/quicklisp/local-projects/
(ql:quickload :cl-webdriver-client)
```

You need a running instance of selenium-server-standalone.

[Download](<http://www.seleniumhq.org/download/>) it and run:

```
curl -LO https://goo.gl/SP94ZB -o selenium-server-standalone.jar
java -jar selenium-server-standalone.jar
```

4 Utils

There is a `:webdriver-client-utils` package which should reduce boilerplate. For example:

```
(defpackage my-test
  (:use :cl :webdriver-client)
  (:import-from :webdriver-client-utils
    :send-keys
    :click
    :wait-for
    :classlist))

(in-package :my-test)

(with-session ()
  (setf (url) "http://google.com")
  (send-keys "cl-webdriver-client")
  (click "[name=btnK]")
  (wait-for "#resultStats"))
```

4.1 Interactive session

You can just start the session and control it from your repl:

```
(in-package :my-test)

(start-interactive-session)

(setf (url) "http://google.com")
(send-keys "cl-webdriver-client")
(send-keys (key :enter))
(classlist "#slim_appbar") ; prints ("ab_tnav_wrp")

(stop-interactive-session)
```

4.2 Utils API conventions

If utility function needs an element to work on it defaults to `‘(active-element)’`.

```
(click) ; click on the current active element.
```

You can also pass a css selector as a last parameter.

```
(print (id "#submit")) ; print id the of matched element
```

```
(assert (= (first (classlist "div")) "first-div-ever"))
```

To change default element you can:

```
(setf webdriver-client-utils:*default-element-func* (lambda () (find-element "input[ty
```

4.3 Waiting for the reaction

Often you need to wait for some action to be done. For example if you do a `(click)` on the button to load search results, you need to wait them to load.

```
(wait-for ".search-result" :timeout 10) ; wait 10 seconds
```

Timeout defaults to 30 seconds. You can globally change it:

```
(setf webdriver-client-utils:*timeout* 3)
```

4.4 Running tests

REPL

```
(ql:quickload '(:cl-selenium :prove))  
(setf prove:*enable-colors* nil)  
(prove:run :cl-webdriver-client-test)
```

Shell

```
sh  
./test.sh
```

5 API

5.1 WEBDRIVER-CLIENT package

WEBDRIVER-CLIENT [PACKAGE]

This package exports functions for working with Selenium WebDriver.

For documentation see:

- <https://github.com/SeleniumHQ/selenium/wiki/JsonWireProtocol>
- <https://www.w3.org/TR/webdriver1>

External definitions

Uncategorized

LOGS (*type* &key (*session* **session**)) [WEBDRIVER-CLIENT]

Return the logs of a particular *TYPE*.

See: [LOG-TYPES], page 10.

NO-SUCH-ELEMENT-ERROR [WEBDRIVER-CLIENT]

Error signaled when no such element is found.

Class precedence list: `no-such-element-error`, `find-error`, `error`, `serious-condition`, `condition`, `t`

MOUSE-MOVE-TO (*x y* &key *element* (*session* **session**)) [WEBDRIVER-CLIENT]

Move the mouse by an offset of the specified *element*. If no *element* is specified, the move is relative to the current mouse cursor. If an *element* is provided but no offset, the mouse will be moved to the center of the *element*. If the *element* is not visible, it will be scrolled into view.

See: <https://github.com/SeleniumHQ/selenium/wiki/JsonWireProtocol#sessionsessionidmoveto>

ELEMENT-ID (*sb-pcl::object*) [WEBDRIVER-CLIENT]

MOUSE-CLICK (*button* &key (*session* **session**)) [WEBDRIVER-CLIENT]

Click any mouse *button* (at the coordinates set by the last `moveto` command). Note that calling this command after calling `buttondown` and before calling *button* up (or any out-of-order interactions sequence) will yield undefined behaviour).

See: <https://github.com/SeleniumHQ/selenium/wiki/JsonWireProtocol#sessionsessionidclick>

LOG-TYPES (&key (*session* **session**)) [WEBDRIVER-CLIENT]

Return the types of logs supported by the WebDriver.

- browser: Javascript console logs from the browser.
- client: Logs from the client side implementation of the WebDriver protocol (e.g. the

Java bindings).

- driver: Logs from the internals of the driver (e.g. FirefoxDriver internals).
- performance: Logs relating to the performance characteristics of the page under test (e.g. resource load timings).
- server: Logs from within the selenium server.

See: <https://github.com/SeleniumHQ/selenium/wiki/Logging> .

MAKE-CAPABILITIES (**&key** (*(:always-match* [WEBDRIVER-CLIENT]
#:always-match) *nil*) (*(:first-match* *#:first-match*) *nil*))

WEBDRIVER-STATUS *nil* [WEBDRIVER-CLIENT]
 Get WebDriver status information

Navigation

BACK (**&key** (*session* **session**)) [WEBDRIVER-CLIENT]
 This command causes the browser to traverse one step backward in the joint *session* history of the current top-level browsing context. This is equivalent to pressing the back button in the browser chrome or invoking `window.history.back`.

Category: Navigation

See: <https://www.w3.org/TR/webdriver1/#dfn-back> .

PAGE-SOURCE (**&key** (*session* **session**)) [WEBDRIVER-CLIENT]
 Returns a string serialization of the DOM of the current browsing context active document.

Category: Navigation

See: <https://www.w3.org/TR/webdriver1/#get-page-source>

PAGE-TITLE (**&key** (*session* **session**)) [WEBDRIVER-CLIENT]
 This command returns the document title of the current top-level browsing context, equivalent to calling `document.title`.

Category: Navigation

See: <https://www.w3.org/TR/webdriver2/#get-title> .

REFRESH (**&key** (*session* **session**)) [WEBDRIVER-CLIENT]
 Refresh the current page.

Category: Navigation

SWITCH-TO-FRAME (*id* **&key** (*session* **session**)) [WEBDRIVER-CLIENT]
 Change focus to another frame on the page. If the frame *id* is null, the server should switch to the page's default content.

In the context of a web browser, a frame is a part of a web page or browser window which displays content independent of its container, with the ability to load content

independently.

Category: Navigation

See: <https://github.com/SeleniumHQ/selenium/wiki/JsonWireProtocol#sessionidframe> .

See: [https://en.wikipedia.org/wiki/Frame_\(World_Wide_Web\)](https://en.wikipedia.org/wiki/Frame_(World_Wide_Web)) .

URL (**&key** (*session* **session**)) [WEBDRIVER-CLIENT]
Get the current url in *session*.

Category: Navigation

See: <https://www.w3.org/TR/webdriver1/#dfn-get-current-url> .

Session

START-INTERACTIVE-SESSION (**&optional** *capabilities*) [WEBDRIVER-CLIENT]
Start an interactive session. Use this to interact with Selenium driver from a REPL.

Category: Session

See: [MAKE-SESSION], page 12

WITH-SESSION (*capabilities* **&body** *body*) [WEBDRIVER-CLIENT]
Execute *BODY* inside a Selenium session.

Category: Session

See: [MAKE-SESSION], page 12

USE-SESSION (*session*) [WEBDRIVER-CLIENT]
Make *SESSION* the current *session*.

Category: Session

STOP-INTERACTIVE-SESSION *nil* [WEBDRIVER-CLIENT]
Stop an interactive session.

Category: Session

MAKE-SESSION (*capabilities*) [WEBDRIVER-CLIENT]
Creates a new WebDriver session with the endpoint node. If the creation fails, a session not created error is returned.

CAPABILITIES are the *capabilities* to negotiate for the new session. If it is NIL, then [*DEFAULT-CAPABILITIES*], page 15 are used. If it is a list, then it is use as parameters for [MAKE-CAPABILITIES], page 11 to build a new *CAPABILITIES* object. Otherwise, it is assumed to be a *CAPABILITIES* object.

Category: Session

See: <https://www.w3.org/TR/webdriver1/#new-session> .

See: <https://www.w3.org/TR/webdriver1/#capabilities> .

DELETE-SESSION (*session*) [WEBDRIVER-CLIENT]
Delete the WebDriver *SESSION*.

Category: *Session*

Screen capture

SCREENSHOT (**&key** (*session* **session**)) [WEBDRIVER-CLIENT]
Screenshots are a mechanism for providing additional visual diagnostic information. They work by dumping a snapshot of the initial viewport's framebuffer as a lossless PNG image. It is returned to the local end as a Base64 encoded string.

Category: Screen capture

See: <https://www.w3.org/TR/webdriver2/#screen-capture> .

ELEMENT-SCREENSHOT (*element* **&key** (*session* **session**)) [WEBDRIVER-CLIENT]

The Take *Element* Screenshot command takes a screenshot of the visible region encompassed by the bounding rectangle of an *element*. If given a parameter argument *scroll* that evaluates to false, the *element* will not be scrolled into view.

Category: Screen capture

See: <https://www.w3.org/TR/webdriver1/#take-element-screenshot> .

Actions

KEY (*key*) [WEBDRIVER-CLIENT]
Returns a string with *KEY*'s codepoint.

Category: Actions

See: <https://www.w3.org/TR/webdriver/#keyboard-actions>

PERFORM-ACTIONS (*actions* **&optional** (*session* **session**)) [WEBDRIVER-CLIENT]

The *Actions* API provides a low-level interface for providing virtualised device input to the web browser.

Conceptually, the *Actions* commands divide time into a series of ticks. The local end sends a series of *actions* which correspond to the change in state, if any, of each input device during each tick. For example, pressing a key is represented by an action sequence consisting of a single key input device and two ticks, the first containing a keyDown action, and the second a keyUp action, whereas a pinch-zoom input is represented by an action sequence consisting of three ticks and two pointer input devices of type touch, each performing a sequence of *actions* pointerDown, followed by pointerMove, and then pointerUp.

Category: *Actions*

See: <https://www.w3.org/TR/webdriver/#actions>

User prompts

ACCEPT-ALERT (**&key** (*session* **session**)) [WEBDRIVER-CLIENT]
Accept Alert.

Category: User prompts

See: <https://www.w3.org/TR/webdriver1/#dfn-accept-alert>

ALERT-TEXT (**&key** (*session* **session**)) [WEBDRIVER-CLIENT]
Get Alert Text.

Category: User prompts

See: <https://www.w3.org/TR/webdriver1/#get-alert-text>

DISMISS-ALERT (**&key** (*session* **session**)) [WEBDRIVER-CLIENT]
The Dismiss Alert command dismisses a simple dialog if present. A request to dismiss an alert user prompt, which may not necessarily have a dismiss button, has the same effect as accepting it.

Category: User prompts

See: <https://www.w3.org/TR/webdriver1/#dismiss-alert>

Document handling

EXECUTE-SCRIPT (*script args* **&key** (*session* **session**)) [WEBDRIVER-CLIENT]
Inject a snippet of JavaScript into the page for execution in the context of the currently selected frame. The executed *script* is assumed to be synchronous and the result of evaluating the *script* is returned to the client.

The *script* argument defines the *script* to execute in the form of a function body. The value returned by that function will be returned to the client. The function will be invoked with the provided *args* array and the values may be accessed via the *arguments* object in the order specified.

Arguments may be any JSON-primitive, array, or JSON object. JSON objects that define a *WebElement* reference will be converted to the corresponding DOM element. Likewise, any *WebElements* in the *script* result will be returned to the client as *WebElement* JSON objects.

Category: Document handling

See: <https://www.w3.org/TR/webdriver1/#executing-script> .

Capabilities

MERGE-CAPABILITIES (*cap1 cap2*) [WEBDRIVER-CLIENT]
Merge two capabilities

Category: Capabilities

CHROME-CAPABILITIES (*&rest options*) [WEBDRIVER-CLIENT]
 Specifies Chrome specific capabilities.

Category: Capabilities

See: https://chromedriver.chromium.org/capabilities#h.p_ID_102

FIREFOX-CAPABILITIES (*&rest options*) [WEBDRIVER-CLIENT]
 Specify capabilities for Firefox browser.

Example usage:

(firefox-capabilities :args #("-headless"))

Category: Capabilities

See: <https://developer.mozilla.org/en-US/docs/Web/WebDriver/Capabilities/firefoxOptions>

DEFAULT-CAPABILITIES [WEBDRIVER-CLIENT]
 The default capabilities.

Category: Capabilities

Cookies

DELETE-ALL-COOKIES (*&key (session *session*)*) [WEBDRIVER-CLIENT]
 Deletes all cookies

Category: Cookies

See: <https://www.w3.org/TR/webdriver1/#delete-all-cookies>

MAKE-COOKIE (*name value &key path domain secure expiry*) [WEBDRIVER-CLIENT]
 Create a cookie object.

Category: Cookies

DELETE-COOKIE (*cookie-name &key (session *session*)*) [WEBDRIVER-CLIENT]
 Delete the cookie with name *COOKIE-NAME*.

Category: Cookies

See: <https://www.w3.org/TR/webdriver1/#delete-cookie>

FIND-COOKIE (*cookie-name &key (session *session*)*) [WEBDRIVER-CLIENT]
 Retrieve the cookie with name *COOKIE-NAME*.

Category: Cookies

See: <https://www.w3.org/TR/webdriver1/#get-named-cookie>

COOKIE [WEBDRIVER-CLIENT]

A cookie is described in [RFC6265] by a name-value pair holding the cookie's data, followed by zero or more attribute-value pairs describing its characteristics.

Category: Cookies

Class precedence list: `cookie`, `standard-object`, `t`

Slots:

- **name** — initarg: `:name`
The name of the cookie
- **value** — initarg: `:value`
The cookie value
- **path** — initarg: `:path`
The cookie path. Defaults to `'/'` if omitted when adding a cookie.
- **domain** — initarg: `:domain`
The domain the cookie is visible to. Defaults to the current browsing context's active document's URL domain if omitted when adding a cookie.
- **secure** — initarg: `:secure`
Whether the cookie is a secure cookie. Defaults to false if omitted when adding a cookie.
- **http-only** — initarg: `:http-only`
Whether the cookie is an HTTP only cookie. Defaults to false if omitted when adding a cookie.
- **expiry** — initarg: `:expiry`
When the cookie expires, specified in seconds since Unix Epoch. Must not be set if omitted when adding a cookie.

COOKIE (&key (session *session*)) [WEBDRIVER-CLIENT]

Retrieve all cookies visible to the current page.

Category: Cookies

See: <https://www.w3.org/TR/webdriver1/#get-all-cookies> .

See: <https://github.com/SeleniumHQ/selenium/wiki/JsonWireProtocol#sessionsessionidcookie> .

Elements**ACTIVE-ELEMENT (&key (session *session*))** [WEBDRIVER-CLIENT]

Return the active element of the current browsing context's document.

The active element is the Element within the DOM that currently has focus.

If there's no active element, an error is signaled.

Category: Elements

See: <https://www.w3.org/TR/webdriver2/#get-active-element> .

See: <https://developer.mozilla.org/en-US/docs/Web/API/Document/activeElement> .

FIND-ELEMENTS (*value* **&key** (*by :css-selector*) (*session* **[WEBDRIVER-CLIENT]**
session))

Find elements that match *VALUE* using location strategy in *BY*.

Category: Elements

See [FIND-ELEMENT], page 18.

See <https://www.w3.org/TR/webdriver1/#find-elements> .

ELEMENT-DISPLAYED (*element* **&key** (*session* **[WEBDRIVER-CLIENT]**
session))

Returns if *ELEMENT* is visible.

Category: Elements

See: <https://www.w3.org/TR/webdriver1/#element-displayedness> .

ELEMENT-ATTRIBUTE (*element name* **&key** (*session* **[WEBDRIVER-CLIENT]**
session))

Return the *ELEMENT*'s attribute named *NAME*.

Category: Elements

ELEMENT-ENABLED (*element* **&key** (*session* **[WEBDRIVER-CLIENT]**
session))

Returns if *ELEMENT* is enabled.

Category: Elements

See: <https://www.w3.org/TR/webdriver1/#is-element-enabled> .

ELEMENT-RECT (*element* **&key** (*session* **[WEBDRIVER-CLIENT]**
session))

The Get *Element* Rect command returns the dimensions and coordinates of the given web *element*. The returned value is a dictionary with the following members:

x

X axis position of the top-left corner of the web *element* relative to the current browsing context's document *element* in CSS pixels.

y

Y axis position of the top-left corner of the web *element* relative to the current browsing context's document *element* in CSS pixels.

height

Height of the web *element*'s bounding rectangle in CSS pixels.

width

Width of the web *element*'s bounding rectangle in CSS pixels.

Category: Elements

ELEMENT-TEXT (*element* **&key** (*session* **[WEBDRIVER-CLIENT]**
session))

The Get *Element* Text command intends to return an *element*'s text "as rendered". An *element*'s rendered text is also used for locating a elements by their link text and partial link text.

Category: Elements

See: <https://www.w3.org/TR/webdriver1/#get-element-text> .

ELEMENT-TAGNAME (*element* &**key** (*session* **session**)) [WEBDRIVER-CLIENT]
Return the *ELEMENT*'s tag name.

Category: Elements

FIND-ELEMENT (*value* &**key** (*by* :*css-selector*) (*session* **session**)) [WEBDRIVER-CLIENT]
The Find Element command is used to find an element in the current browsing context that can be used as the web element context for future element-centric commands.

For example, consider this pseudo code which retrieves an element with the #toremove ID and uses this as the argument for a script it injects to remove it from the HTML document:

```
let body <undefined> [=], page <undefined> session.find.css("#toremove");
session.execute("arguments[0].remove()", [body]);
```

The *BY* parameter represents the element location strategy.

It can be one of:

- *:id* : Finds element *by* id.
- *:class-name* : Finds element *by* class name.
- *:css-selector* : Returns element that matches css selector.
- *:link-text* : Returns element that matches <a> element text.
- *:partial-link-text*: Returns element that matches <a> element text partially.
- *:tag-name*: Returns element that matches tag name.
- *:xpath*: Returns element that matches the XPath expression.

If result is empty, a <undefined> [HANDLE-FIND-ERROR], page <undefined> is signaled.

Category: Elements

See: <https://www.w3.org/TR/webdriver1/#dfn-find-element> .

Element interaction

ELEMENT-CLEAR (*element* &**key** (*session* **session**)) [WEBDRIVER-CLIENT]
Clear the contents of *ELEMENT* (for example, a form field *element*).

Category: *Element* interaction

See: <https://www.w3.org/TR/webdriver1/#dfn-element-clear> .

ELEMENT-CLICK (*element* &**key** (*session* **session**)) [WEBDRIVER-CLIENT]
The *Element* Click command scrolls into view the *element* if it is not already pointer-interactable, and clicks its in-view center point.

If the *element*'s center point is obscured by another *element*, an *element* click intercepted error is returned. If the *element* is outside the viewport, an *element* not interactable error is returned.

Category: *Element* interaction

See: <https://www.w3.org/TR/webdriver1/#element-click> .

ELEMENT-SEND-KEYS (*element* *keys* **&key** (*session* [WEBDRIVER-CLIENT] **session**))

The *Element* Send Keys command scrolls into view the form control *element* and then sends the provided *keys* to the *element*. In case the *element* is not keyboard-interactable, an *element* not interactable error is returned.

KEYS should be a string.

Category: *Element* interaction

See: <https://www.w3.org/TR/webdriver1/#element-send-keys> .

Windows

CLOSE-CURRENT-WINDOW (**&key** (*session* **session**)) [WEBDRIVER-CLIENT]

Close the current window.

Category: Windows

5.2 WEBDRIVER-CLIENT-UTILS package

WEBDRIVER-CLIENT-UTILS [PACKAGE]

Package with the purpose of reducing boilerplate.

The exported definitions work with an implicit element. The default implicit element is the current active element. So, it is not necessary to pass the element you are working with around most of the time.

External definitions

Variables

TIMEOUT [WEBDRIVER-CLIENT-UTILS]

Default timeout value to use in selenium-utils functions.

DEFAULT-ELEMENT-FUNC [WEBDRIVER-CLIENT-UTILS]

Function used to get the 'default element' by selenium-utils functions.

It is [ACTIVE-ELEMENT], page 16 function by default.

Functions

ID (&optional <i>selector</i>)	[WEBDRIVER-CLIENT-UTILS]
Get active element id.	
GET-COOKIE (<i>cookie name</i>)	[WEBDRIVER-CLIENT-UTILS]
Get value of <i>COOKIE</i> at <i>NAME</i> .	
FIND-ELEM (<i>selector</i> &key (<i>by :css-selector</i>))	[WEBDRIVER-CLIENT-UTILS]
Find element <i>by</i> <i>SELECTOR</i> . Returns NIL if the element is not found.	
WAIT-FOR (<i>selector</i> &key (<i>timeout *timeout*</i>))	[WEBDRIVER-CLIENT-UTILS]
Wait for an element that matches <i>SELECTOR</i> to appear on the screen.	
<i>TIMEOUT</i> indicates how much time to wait (default is <i>*TIMEOUT*</i>).	
CLASSNAME (&optional <i>selector</i>)	[WEBDRIVER-CLIENT-UTILS]
Get active element classname.	
TEXT (&optional <i>selector</i>)	[WEBDRIVER-CLIENT-UTILS]
Get active element's text.	
SEND-KEY (<i>key</i> &optional <i>selector</i>)	[WEBDRIVER-CLIENT-UTILS]
Send a <i>key</i> to active element.	
CLASSLIST (&optional <i>selector</i>)	[WEBDRIVER-CLIENT-UTILS]
Get active element class list.	
ATTR (<i>name</i> &optional <i>selector</i>)	[WEBDRIVER-CLIENT-UTILS]
Get active element attribute.	
SEND-KEYS (<i>keys</i> &optional <i>selector</i>)	[WEBDRIVER-CLIENT-UTILS]
Send <i>keys</i> to active element.	
ELEM (&optional <i>selector</i>)	[WEBDRIVER-CLIENT-UTILS]
If <i>SELECTOR</i> is given, wait for an element that matches the <i>selector</i> to appear.	
Otherwise, call [*DEFAULT-ELEMENT-FUNC*], page 19 (the active element is returned by default).	
CLICK (&optional <i>selector</i>)	[WEBDRIVER-CLIENT-UTILS]
Click on active element.	

6 Index

(Index is nonexistent)

*

DEFAULT-CAPABILITIES	15
DEFAULT-ELEMENT-FUNC	19
TIMEOUT	19

A

ACCEPT-ALERT	14
ACTIVE-ELEMENT	16
ALERT-TEXT	14
ATTR	20

B

BACK	11
------	----

C

CHROME-CAPABILITIES	15
CLASSLIST	20
CLASSNAME	20
CLICK	20
CLOSE-CURRENT-WINDOW	19
COOKIE	16

D

DELETE-ALL-COOKIES	15
DELETE-COOKIE	15
DELETE-SESSION	13
DISMISS-ALERT	14

E

ELEM	20
ELEMENT-ATTRIBUTE	17
ELEMENT-CLEAR	18
ELEMENT-CLICK	18
ELEMENT-DISPLAYED	17
ELEMENT-ENABLED	17
ELEMENT-ID	10
ELEMENT-RECT	17
ELEMENT-SCREENSHOT	13
ELEMENT-SEND-KEYS	19
ELEMENT-TAGNAME	18
ELEMENT-TEXT	17
EXECUTE-SCRIPT	14

W

WEBDRIVER-CLIENT-	
UTILS:*DEFAULT-ELEMENT-FUNC*	19
WEBDRIVER-CLIENT-UTILS:*TIMEOUT*	19
WEBDRIVER-CLIENT:*DEFAULT-CAPABILITIES*	15

F

FIND-COOKIE	15
FIND-ELEM	20
FIND-ELEMENT	18
FIND-ELEMENTS	17
FIREFOX-CAPABILITIES	15

G

GET-COOKIE	20
------------	----

I

ID	20
----	----

K

KEY	13
-----	----

L

LOG-TYPES	10
LOGS	10

M

MAKE-CAPABILITIES	11
MAKE-COOKIE	15
MAKE-SESSION	12
MERGE-CAPABILITIES	14
MOUSE-CLICK	10
MOUSE-MOVE-TO	10

P

PAGE-SOURCE	11
PAGE-TITLE	11
PERFORM-ACTIONS	13

R

REFRESH	11
---------	----

S

SCREENSHOT	13
SEND-KEY	20
SEND-KEYS	20
START-INTERACTIVE-SESSION	12
STOP-INTERACTIVE-SESSION	12
SWITCH-TO-FRAME	11

T

TEXT	20
------------	----

U

URL	12
USE-SESSION	12

W

WAIT-FOR	20
WEBDRIVER-CLIENT-UTILS:ATTR	20
WEBDRIVER-CLIENT-UTILS:CLASSLIST	20
WEBDRIVER-CLIENT-UTILS:CLASSNAME	20
WEBDRIVER-CLIENT-UTILS:CLICK	20
WEBDRIVER-CLIENT-UTILS:ELEM	20
WEBDRIVER-CLIENT-UTILS:FIND-ELEM	20
WEBDRIVER-CLIENT-UTILS:GET-COOKIE	20
WEBDRIVER-CLIENT-UTILS:ID	20
WEBDRIVER-CLIENT-UTILS:SEND-KEY	20
WEBDRIVER-CLIENT-UTILS:SEND-KEYS	20
WEBDRIVER-CLIENT-UTILS:TEXT	20
WEBDRIVER-CLIENT-UTILS:WAIT-FOR	20
WEBDRIVER-CLIENT:ACCEPT-ALERT	14
WEBDRIVER-CLIENT:ACTIVE-ELEMENT	16
WEBDRIVER-CLIENT:ALERT-TEXT	14
WEBDRIVER-CLIENT:BACK	11
WEBDRIVER-CLIENT:CHROME-CAPABILITIES	15
WEBDRIVER-CLIENT:CLOSE-CURRENT-WINDOW	19
WEBDRIVER-CLIENT:COOKIE	16
WEBDRIVER-CLIENT:DELETE-ALL-COOKIES	15
WEBDRIVER-CLIENT:DELETE-COOKIE	15
WEBDRIVER-CLIENT:DELETE-SESSION	13

WEBDRIVER-CLIENT:DISMISS-ALERT	14
WEBDRIVER-CLIENT:ELEMENT-ATTRIBUTE	17
WEBDRIVER-CLIENT:ELEMENT-CLEAR	18
WEBDRIVER-CLIENT:ELEMENT-CLICK	18
WEBDRIVER-CLIENT:ELEMENT-DISPLAYED	17
WEBDRIVER-CLIENT:ELEMENT-ENABLED	17
WEBDRIVER-CLIENT:ELEMENT-ID	10
WEBDRIVER-CLIENT:ELEMENT-RECT	17
WEBDRIVER-CLIENT:ELEMENT-SCREENSHOT	13
WEBDRIVER-CLIENT:ELEMENT-SEND-KEYS	19
WEBDRIVER-CLIENT:ELEMENT-TAGNAME	18
WEBDRIVER-CLIENT:ELEMENT-TEXT	17
WEBDRIVER-CLIENT:EXECUTE-SCRIPT	14
WEBDRIVER-CLIENT:FIND-COOKIE	15
WEBDRIVER-CLIENT:FIND-ELEMENT	18
WEBDRIVER-CLIENT:FIND-ELEMENTS	17
WEBDRIVER-CLIENT:FIREFOX-CAPABILITIES	15
WEBDRIVER-CLIENT:KEY	13
WEBDRIVER-CLIENT:LOG-TYPES	10
WEBDRIVER-CLIENT:LOGS	10
WEBDRIVER-CLIENT:MAKE-CAPABILITIES	11
WEBDRIVER-CLIENT:MAKE-COOKIE	15
WEBDRIVER-CLIENT:MAKE-SESSION	12
WEBDRIVER-CLIENT:MERGE-CAPABILITIES	14
WEBDRIVER-CLIENT:MOUSE-CLICK	10
WEBDRIVER-CLIENT:MOUSE-MOVE-TO	10
WEBDRIVER-CLIENT:PAGE-SOURCE	11
WEBDRIVER-CLIENT:PAGE-TITLE	11
WEBDRIVER-CLIENT:PERFORM-ACTIONS	13
WEBDRIVER-CLIENT:REFRESH	11
WEBDRIVER-CLIENT:SCREENSHOT	13
WEBDRIVER-CLIENT:START- INTERACTIVE-SESSION	12
WEBDRIVER-CLIENT:STOP- INTERACTIVE-SESSION	12
WEBDRIVER-CLIENT:SWITCH-TO-FRAME	11
WEBDRIVER-CLIENT:URL	12
WEBDRIVER-CLIENT:USE-SESSION	12
WEBDRIVER-CLIENT:WEBDRIVER-STATUS	11
WEBDRIVER-CLIENT:WITH-SESSION	12
WEBDRIVER-STATUS	11
WITH-SESSION	12