Package 'selenium'

November 28, 2023

Title Low-Level Browser Automation Interface

Version 0.1.3

Description An implementation of 'W3C WebDriver 2.0'
(<https://w3c.github.io/webdriver/>), allowing interaction
with a 'Selenium Server' (<https://www.selenium.dev/documentation/grid/>)
instance from 'R'. Allows a web browser to be automated from 'R'.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Depends R (>= 2.10)

Suggests gitcreds, testthat (>= 3.0.0), withr, xml2

Config/testthat/edition 3

Imports httr2, jsonlite, lifecycle, processx, R6, rappdirs, rlang (>=
1.1.0)

URL <https://ashbythorpe.github.io/selenium-r/>,
<https://github.com/ashbythorpe/selenium-r>

Config/Needs/website rmarkdown

BugReports <https://github.com/ashbythorpe/selenium-r/issues>

Language en-GB

NeedsCompilation no

Author Ashby Thorpe [aut, cre, cph] (<https://orcid.org/0000-0003-3106-099X>)

Maintainer Ashby Thorpe <ashbythorpe@gmail.com>

Repository CRAN

Date/Publication 2023-11-28 18:40:02 UTC

R **topics documented:**

Index   **[69](#)**

---

actions_mousedown	*Press, release or move the mouse*

Description

Mouse actions to be passed into [actions_stream()](#). actions_mousedown() represents pressing a
button on the mouse, while actions_mouseup() represents releasing a button. actions_mousemove()
represents moving the mouse.

Usage

```
actions_mousedown(
  button = c("left", "right", "middle"),
  width = NULL,
  height = NULL,
  pressure = NULL,
  tangential_pressure = NULL,
  tilt_x = NULL,
  tilt_y = NULL,
  twist = NULL,
  altitude_angle = NULL,
  azimuth_angle = NULL
)

actions_mouseup(
  button = c("left", "right", "middle"),
  width = NULL,
  height = NULL,
  pressure = NULL,
  tangential_pressure = NULL,
  tilt_x = NULL,
```

```
    tilt_y = NULL,
    twist = NULL,
    altitude_angle = NULL,
    azimuth_angle = NULL
)

actions_mousemove(
  x,
  y,
  duration = NULL,
  origin = c("viewport", "pointer"),
  width = NULL,
  height = NULL,
  pressure = NULL,
  tangential_pressure = NULL,
  tilt_x = NULL,
  tilt_y = NULL,
  twist = NULL,
  altitude_angle = NULL,
  azimuth_angle = NULL
)
```

Arguments

`button`	The mouse button to press.
`width`	The 'width' of the click, a number.
`height`	The 'height' of the click, a number.
`pressure`	The amount of pressure to apply to the click: a number between 0 and 1.
`tangential_pressure`	
	A number between 0 and 1.
`tilt_x`	A whole number between -90 and 90.
`tilt_y`	A whole number between -90 and 90.
`twist`	A whole number between 0 and 359.
`altitude_angle`	A number between 0 and `pi/2`.
`azimuth_angle`	A number between 0 and `2*pi`.
`x`	The x coordinate of the mouse movement.
`y`	The y coordinate of the mouse movement.
`duration`	The duration of the mouse movement, in seconds.
`origin`	The point from which x and y are measured. Can be a `WebElement` object, in which case x and y are measured from the center of the element.

Value

A `selenium_action` object.

Examples

```
actions_stream(
  actions_mousedown("left", width = 1, height = 1, pressure = 0.5),
  actions_mouseup("left", width = 100, height = 50, pressure = 1),
  actions_mousemove(x = 1, y = 1, duration = 1, origin = "pointer")
)
```

---

actions_pause	*Wait for a period of time*

Description

A pause action to be passed into [actions_stream()](#). Waits for a given number of seconds before performing the next action in the stream.

Usage

```
actions_pause(seconds)
```

Arguments

seconds	The number of seconds to wait for.

Value

A `selenium_action` object.

Examples

```
actions_stream(
  actions_pause(1)
)
```

---

actions_press	*Press or release a key*

Description

Key actions to be passed into [actions_stream()](#). actions_press() represents pressing a key on the keyboard, while actions_release() represents releasing a key.

Usage

```
actions_press(key)

actions_release(key)
```

Arguments

key	The key to press: a string consisting of a single character. Use the [keys](#) object to use special keys (e.g. `Ctrl`).

Value

A `selenium_action` object.

Examples

```
actions_stream(
  actions_press("a"),
  actions_release("a"),
  actions_press(keys$enter),
  actions_release(keys$enter)
)
```

---

actions_scroll   *Scroll the page*

Description

Scroll actions to be passed into [`actions_stream()`](#). Scroll the page in a given direction.

Usage

```
actions_scroll(x, y, delta_x, delta_y, duration = NULL, origin = "viewport")
```

Arguments

x	The x coordinate from which the scroll action originates from.
y	The y coordinate from which the scroll action originates from.
delta_x	The number of pixels to scroll in the x direction.
delta_y	The number of pixels to scroll in the y direction.
duration	The duration of the scroll, in seconds.
origin	The point from which x and y are measured. Can be a `WebElement` object, in which case x and y are measured from the center of the element. Otherwise, `origin` must be `"viewport"`.

Value

A `selenium_action` object.

Examples

```
actions_stream(
  actions_scroll(x = 1, y = 1, delta_x = 1, delta_y = 1, duration = 0.5)
)
```

---

actions_stream	*Create a set of actions to be performed*

Description

actions_stream() creates a set of actions to be performed by SeleniumSession$perform_actions().
Actions are a low level way to interact with a page.

Usage

```
actions_stream(...)
```

Arguments

...   selenium_action objects: the actions to perform.

Value

A selenium_actions_stream object, ready to be passed into SeleniumSession$perform_actions().

See Also

- Pause actions: actions_pause().

- Press actions: actions_press() and actions_release().

- Mouse actions: actions_mousedown(), actions_mouseup() and actions_mousemove().

- Scroll actions: actions_scroll().

Examples

```
actions_stream(
  actions_press(keys$enter),
  actions_pause(0.5),
  actions_release(keys$enter),
  actions_scroll(x = 1, y = 1, delta_x = 1, delta_y = 1, duration = 0.5),
  actions_mousemove(x = 1, y = 1, duration = 1, origin = "pointer")
)
```

---

keys *A list of special keys*

Description

A named list of special keys, where each key is a single Unicode character, which will be interpreted by selenium as a special key. Each key is just a string, so can be used with string manipulation functions like [paste()](#) without any special treatment.

Usage

```
keys
```

Format

An object of class list of length 65.

Examples

```
keys$enter
```

---

key_chord *Combine special keys*

Description

When a chord of keys is passed into WebElement$send_keys(), all keys will be pressed in order, and then released at the end. This is simply done by combining the keys into a single string, and appending the NULL key ([keys$null](#)) to the end. This is useful for keybindings like Ctrl-V, where you want the Ctrl key to be released after the action.

Usage

```
key_chord(...)
```

Arguments

... The keys to be combined (strings).

Value

A string.

Examples

```
# `Ctrl-V` will be pressed, then `Ctrl-Alt-V`
paste0(
  keys$control, "v",
  keys$alt, "v"
)

# `Ctrl-V` will be pressed, then `Alt-V`
paste0(
  key_chord(keys$control, "v"),
  key_chord(keys$alt, "v")
)
```

SeleniumSession   *Start a Selenium Client session*

Description

This class represents the client to a Selenium session. It will only work if a Selenium server instance
is running. If you get an error, use selenium_server_available() to check if a server is running.
See the package README for more information, or use selenium_server() to try and start a
server automatically.

Public fields

id The id of the session, generated when the session is started.

browser The browser that the session is using.

port The port that the session is using.

host The host that the session is running on.

Methods

Public methods:

- SeleniumSession$new()
- SeleniumSession$create_webelement()
- SeleniumSession$create_shadowroot()
- SeleniumSession$close()
- SeleniumSession$status()
- SeleniumSession$get_timeouts()
- SeleniumSession$set_timeouts()
- SeleniumSession$navigate()
- SeleniumSession$current_url()
- SeleniumSession$back()
- SeleniumSession$forward()

- SeleniumSession$refresh()
- SeleniumSession$title()
- SeleniumSession$window_handle()
- SeleniumSession$close_window()
- SeleniumSession$switch_to_window()
- SeleniumSession$window_handles()
- SeleniumSession$new_window()
- SeleniumSession$switch_to_frame()
- SeleniumSession$switch_to_parent_frame()
- SeleniumSession$get_window_rect()
- SeleniumSession$set_window_rect()
- SeleniumSession$maximize_window()
- SeleniumSession$minimize_window()
- SeleniumSession$fullscreen_window()
- SeleniumSession$get_active_element()
- SeleniumSession$find_element()
- SeleniumSession$find_elements()
- SeleniumSession$get_page_source()
- SeleniumSession$execute_script()
- SeleniumSession$execute_async_script()
- SeleniumSession$get_cookies()
- SeleniumSession$get_cookie()
- SeleniumSession$add_cookie()
- SeleniumSession$delete_cookie()
- SeleniumSession$delete_all_cookies()
- SeleniumSession$perform_actions()
- SeleniumSession$release_actions()
- SeleniumSession$dismiss_alert()
- SeleniumSession$accept_alert()
- SeleniumSession$get_alert_text()
- SeleniumSession$send_alert_text()
- SeleniumSession$screenshot()
- SeleniumSession$print_page()
- SeleniumSession$clone()

Method new(): Create a Selenium session: opening a browser which can be controlled by the Selenium client.

Usage:

```
SeleniumSession$new(
  browser = "firefox",
  port = 4444L,
  host = "localhost",
  verbose = FALSE,
```

```
  capabilities = NULL,
  request_body = NULL,
  timeout = 20
)
```

Arguments:

browser The name of the browser to use (e.g. "chrome", "firefox", "edge").

port The port that the Selenium server is using, so we can connect to it.

host The host that the Selenium server is running on. This is usually 'localhost' (i.e. your own machine).

verbose Whether to print the web requests that are being sent and any responses.

capabilities A list of capabilities to pass to the Selenium server, to combine with the defaults generated using browser.

request_body A list of request body parameters to pass to the Selenium server. Overrides capabilities.

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: A SeleniumSession object.

Examples:

```
\dontrun{
session <- SeleniumSession$new(verbose = TRUE)

session$close()
}
```

Method create_webelement(): Create a [WebElement](#) object using the parameters of the current session.

Usage:

```
SeleniumSession$create_webelement(id)
```

Arguments:

id The element id.

Returns: A [WebElement](#) object.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

element <- session$find_element(using = "css selector", value = "*")

element2 <- session$create_webelement(id = element$id)

session$close()
}
```

Method create_shadowroot(): Create a [ShadowRoot](#) object using the parameters of the current session.

Usage:

```
SeleniumSession$create_shadowroot(id)
```

Arguments:

id The shadow root id.

Returns: A [ShadowRoot](#) object.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

shadow_root <- session$create_shadowroot(id = "foo")

session$close()
}
```

Method `close()`: Close the current session. Once a session is closed, its methods will no longer work. However, the Selenium server will still be running.

Usage:

```
SeleniumSession$close(timeout = 20)
```

Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$close()
}
```

Method `status()`: Get the status of the Selenium server. Unlike all other methods, this method is independent of the session itself (meaning it can be used even after [SeleniumSession$close()](#) is called). It is identical to [get_server_status()](#), but uses the host, port and verbose options passed to the session, for convenience.

Usage:

```
SeleniumSession$status(timeout = 20)
```

Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: A list that can (but may not always) contain the following fields:

- `ready`: Whether the server is ready to be connected to. This should always be returned by the server.
- `message`: A message about the status of the server.
- `uptime`: How long the server has been running.
- `nodes`: Information about the slots that the server can take.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$status()

session$close()

session$status()
}
```

Method `get_timeouts()`: Get the timeouts of the current session. There are three types of timeouts:

- *session script timeout*: The amount of time that the server will wait for scripts to run. Defaults to 3 seconds.
- *page load timeout*: The amount of time that the server will wait for the page to load. Defaults to 30 seconds.
- *implicit wait*: The amount of time that the server will wait for elements to be located, or for elements to become interactable when required. Defaults to 0 seconds.

Usage:
```
SeleniumSession$get_timeouts(timeout = 20)
```

Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: A list with three items: `script`, `page_load`, and `implicit`.

Examples:
```
\dontrun{

session <- SeleniumSession$new()

session$get_timeouts()

session$close()
}
```

Method `set_timeouts()`: Set the timeouts of the current session. The types of timeouts are defined in SeleniumSession$get_timeouts().

Usage:
```
SeleniumSession$set_timeouts(
  script = NULL,
  page_load = NULL,
  implicit_wait = NULL,
  request_body = NULL,
  timeout = 20
)
```

Arguments:

`script` The amount of time to wait for scripts. By default, this is not set.

`page_load` The amount of time to wait for the page to load.

`implicit_wait` The amount of time to wait for elements on the page.

`request_body` A list of request body parameters to pass to the Selenium server, overriding the default body of the web request

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$set_timeouts(script = 100)

session$get_timeouts()

session$close()
}
```

Method `navigate()`: Navigate to a URL.

Usage:

```
SeleniumSession$navigate(url, request_body = NULL, timeout = 20)
```

Arguments:

`url` The URL to navigate to. Must begin with a protocol (e.g. 'https://').

`request_body` A list of request body parameters to pass to the Selenium server, overriding the default body of the web request.

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$close()
}
```

Method `current_url()`: Get the current URL.

Usage:

```
SeleniumSession$current_url(timeout = 20)
```

Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: The URL of the current page.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$current_url()

session$close()
}
```

Method back()**:** Go back in the navigation history.

Usage:

```
SeleniumSession$back(timeout = 20)
```

Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$navigate("https://www.tidyverse.org")

session$back()

session$current_url()

session$close()
}
```

Method forward()**:** Go forward in the navigation history.

Usage:

```
SeleniumSession$forward(timeout = 20)
```

Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$navigate("https://www.tidyverse.org")
```

```
session$back()

session$forward()

session$current_url()

session$close()
}
```

Method `refresh()`: Reload the current page.

Usage:
```
SeleniumSession$refresh(timeout = 20)
```

Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$refresh()

session$close()
}
```

Method `title()`: Get the title of the current page.

Usage:
```
SeleniumSession$title(timeout = 20)
```

Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: The title of the current page.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$title()

session$close()
}
```

Method `window_handle()`: Get the current window handle.

Usage:

```
SeleniumSession$window_handle(timeout = 20)
```

Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The handle of the current window (a string).

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$window_handle()

session$close()
}
```

Method `close_window()`: Close the current window.

Usage:
```
SeleniumSession$close_window(timeout = 20)
```

Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$new_window()

session$close_window()

session$close()
}
```

Method `switch_to_window()`: Switch to a specific window.

Usage:
```
SeleniumSession$switch_to_window(handle, request_body = NULL, timeout = 20)
```

Arguments:

handle The handle of the window to switch to.

request_body A list of request body parameters to pass to the Selenium server, overriding the
    default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:
```
\dontrun{
session <- SeleniumSession$new()
```

```
handle <- session$window_handle()

handle2 <- session$new_window()$handle

session$switch_to_window(handle)

session$switch_to_window(handle2)

session$close()
}
```

Method `window_handles()`: Get the handles of all open windows.

Usage:
```
SeleniumSession$window_handles(timeout = 20)
```

Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: The handles of all open windows (a list of strings).

Examples:
```
\dontrun{
session <- SeleniumSession$new()

handles <- session$window_handles()

session$close()
}
```

Method `new_window()`: Create a new window. Note that this window is not automatically switched to.

Usage:
```
SeleniumSession$new_window(
  type = c("tab", "window"),
  request_body = NULL,
  timeout = 20
)
```

Arguments:

`type` Whether to create a tab or a window.

`request_body` A list of request body parameters to pass to the Selenium server, overriding the default body of the web request

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: A list containing two elements:

- `handle`: The handle of the new window.
- `type`: The type of window. ("tab" or "window").

Examples:

```
\dontrun{
session <- SeleniumSession$new()

handle <- session$new_window()$handle

session$switch_to_window(handle)

session$close()
}
```

Method `switch_to_frame()`**:** Frames allow you to split a window into multiple sections, where each section can load a separate HTML document. This function allows you to switch to a specific frame, given its ID, meaning that frame will become the current browsing context.

Usage:

```
SeleniumSession$switch_to_frame(id = NA, request_body = NULL, timeout = 20)
```

Arguments:

id The ID of the frame to switch to. By default, the top-level browsing context is switched to (i.e. not a frame). This can also be a [WebElement](#) object, in which case the frame that contains said element will be switched to.

request_body A list of request body parameters to pass to the Selenium server, overriding the default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$switch_to_frame()

session$close()
}
```

Method `switch_to_parent_frame()`**:** Switch to the parent frame of the current frame.

Usage:

```
SeleniumSession$switch_to_parent_frame(timeout = 20)
```

Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:

```
\dontrun{
session <- SeleniumSession$new()
```

```
session$navigate("https://www.r-project.org")

session$switch_to_frame()

session$switch_to_parent_frame()

session$close()
}
```

Method `get_window_rect()`: Get the size and position of the current window.

Usage:
```
SeleniumSession$get_window_rect(timeout = 20)
```

Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: A list containing four elements:
- `x`: The x position of the window relative to the left of the screen.
- `y`: The y position of the window relative to the top of the screen.
- `width`: The width of the window.
- `height`: The height of the window.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$get_window_rect()

session$close()
}
```

Method `set_window_rect()`: Set the size and position of the current window.

Usage:
```
SeleniumSession$set_window_rect(
  width = NULL,
  height = NULL,
  x = NULL,
  y = NULL,
  request_body = NULL,
  timeout = 20
)
```

Arguments:

`width` The width of the window.

`height` The height of the window.

`x` The x position of the window relative to the left of the screen.

`y` The y position of the window relative to the top of the screen.

`request_body` A list of request body parameters to pass to the Selenium server, overriding the default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$set_window_rect(width = 800, height = 600, x = 2, y = 3)

session$close()
}
```

Method `maximize_window()`: Maximize the current window. This makes the window the maximum size it can be, without being full screen.

Usage:
```
SeleniumSession$maximize_window(timeout = 20)
```

Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$maximize_window()

session$close()
}
```

Method `minimize_window()`: Minimize the current window. This hides the window.

Usage:
```
SeleniumSession$minimize_window(timeout = 20)
```

Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$minimize_window()

session$close()
}
```

Method `fullscreen_window()`: Make the window full screen.

Usage:
```
SeleniumSession$fullscreen_window(timeout = 20)
```

Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$fullscreen_window()

session$close()
}
```

Method `get_active_element()`: Get the currently active element.

Usage:
```
SeleniumSession$get_active_element(timeout = 20)
```

Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: A [WebElement](#) object.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$get_active_element()

session$close()
}
```

Method `find_element()`: Find the first element matching a selector.

Usage:
```
SeleniumSession$find_element(
  using = c("css selector", "xpath", "tag name", "link text", "partial link text"),
  value,
  request_body = NULL,
  timeout = 20
)
```

Arguments:

`using` The type of selector to use.

`value` The value of the selector: a string.

request_body A list of request body parameters to pass to the Selenium server, overriding the default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: A [WebElement](#) object.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "#download")

session$find_element(using = "xpath", value = "//div[contains(@class, 'col-xs')]/h1")

session$close()
}
```

Method find_elements(): Find all elements matching a selector.

Usage:
```
SeleniumSession$find_elements(
  using = c("css selector", "xpath", "tag name", "link text", "partial link text"),
  value,
  request_body = NULL,
  timeout = 20
)
```

Arguments:

using The type of selector to use.

value The value of the selector: a string.

request_body A list of request body parameters to pass to the Selenium server, overriding the default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: A list of [WebElement](#) objects.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_elements(using = "css selector", value = "h1")

session$find_elements(using = "xpath", value = "//h1")

session$close()
}
```

Method get_page_source(): Get the HTML source of the current page, serialized as a string.

Usage:

```
SeleniumSession$get_page_source(timeout = 20)
```

Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: A string.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$get_page_source()

session$close()
}
```

Method execute_script(): Execute a JavaScript script.

Usage:

```
SeleniumSession$execute_script(x, ..., request_body = NULL, timeout = 20)
```

Arguments:

x The script to execute. To return a value, do so explicitly, e.g. return 1.

... Additional arguments to pass to the script. These can be accessed in the script using the arguments array. Can be [WebElement](#) objects or lists of such objects, which will be converted to nodes.

request_body A list of request body parameters to pass to the Selenium server, overriding the default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The return value of the script. Nodes or lists of nodes will be converted to [WebElement](#) objects.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$execute_script("return 1")

session$execute_script("return arguments[0] + arguments[1]", 1, 2)

element <- session$find_element(value = "*")

session$execute_script("return arguments[0]", element)

session$close()
}
```

Method execute_async_script(): Execute an asynchronous JavaScript script, waiting for a value to be returned.

Usage:

```
SeleniumSession$execute_async_script(x, ..., request_body = NULL, timeout = 20)
```

Arguments:

x The script to execute. Unlike `execute_script()`. You return an value using the callback function, which can be accessed using `arguments[arguments.length - 1]`. For example, to return 1, you would write `arguments[arguments.length - 1](1)`. This allows you to write asynchronous JavaScript, but treat it like synchronous R code.

... Additional arguments to pass to the script. Can be [WebElement](#) objects or lists of such objects, which will be converted to nodes.

request_body A list of request body parameters to pass to the Selenium server, overriding the default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The return value of the script. Nodes or lists of nodes will be converted to [WebElement](#) objects.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$execute_async_script("
  let callback = arguments[arguments.length - 1];
  callback(1)
")

session$close()
}
```

Method `get_cookies()`: Get all cookies.

Usage:

```
SeleniumSession$get_cookies(timeout = 20)
```

Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: A list of cookies. Each cookie is a list with a `name` and `value` field, along with some other optional fields.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$get_cookies()

session$close()
}
```

Method `get_cookie()`: Get a specific cookie using its name.

Usage:

```
SeleniumSession$get_cookie(name, request_body = NULL, timeout = 20)
```

Arguments:

name The name of the cookie.

request_body A list of request body parameters to pass to the Selenium server, overriding the default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The cookie object.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$add_cookie(list(name = "foo", value = "bar"))

session$get_cookie("foo")

session$close()
}
```

Method add_cookie(): Add a cookie to the cookie store of the current document.

Usage:

```
SeleniumSession$add_cookie(cookie, request_body = NULL, timeout = 20)
```

Arguments:

cookie The cookie object to add: a list which must contain a name and value field.

request_body A list of request body parameters to pass to the Selenium server, overriding the default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$add_cookie(list(name = "my_cookie", value = "1"))

session$close()
}
```

Method delete_cookie(): Delete a cookie using its name.

Usage:

```
SeleniumSession$delete_cookie(name, request_body = NULL, timeout = 20)
```

Arguments:

name The name of the cookie.

request_body A list of request body parameters to pass to the Selenium server, overriding the default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$add_cookie(list(name = "foo", value = "bar"))

session$delete_cookie("foo")

session$close()
}
```

Method delete_all_cookies(): Delete all cookies in the cookie store of the current document.

Usage:

```
SeleniumSession$delete_all_cookies(timeout = 20)
```

Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$delete_all_cookies()

session$close()
}
```

Method perform_actions(): Perform a sequence of actions.

Usage:

```
SeleniumSession$perform_actions(
  actions,
  release_actions = TRUE,
  request_body = NULL,
  timeout = 20
)
```

Arguments:

actions A selenium_actions_stream object, created using [actions_stream()](#).

release_actions Whether to call release_actions() after performing the actions.

request_body A list of request body parameters to pass to the Selenium server, overriding the default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

actions <- actions_stream(
  actions_press(keys$enter),
  actions_pause(0.5),
  actions_release(keys$enter)
)

session$perform_actions(actions)

session$close()
}
```

Method release_actions()**:** Release all keys and pointers that were pressed using perform_actions().

Usage:

```
SeleniumSession$release_actions(timeout = 20)
```

Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

actions <- actions_stream(
  actions_press("a")
)

session$perform_actions(actions, release_actions = FALSE)

session$release_actions()

session$close()
}
```

Method `dismiss_alert()`: Dismiss the current alert, if present.

Usage:
```
SeleniumSession$dismiss_alert(timeout = 20)
```
Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$execute_script("alert('hello')")

session$dismiss_alert()

session$close()
}
```

Method `accept_alert()`: Accept the current alert, if present.

Usage:
```
SeleniumSession$accept_alert(timeout = 20)
```
Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$execute_script("alert('hello')")

session$accept_alert()

session$close()
}
```

Method `get_alert_text()`: Get the message of the current alert, if present.

Usage:
```
SeleniumSession$get_alert_text(timeout = 20)
```
Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: The message of the current alert (a string).

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$execute_script("alert('hello')")

session$get_alert_text()

session$close()
}
```

Method `send_alert_text()`: Send text to the current alert, if present. Useful if the alert is a prompt.

Usage:

```
SeleniumSession$send_alert_text(text, request_body = NULL, timeout = 20)
```

Arguments:

`text` The text to send.

`request_body` A list of request body parameters to pass to the Selenium server, overriding the default body of the web request

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: The session object, invisibly.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$execute_script("prompt('Enter text:')")

session$send_alert_text("hello")

session$close()
}
```

Method `screenshot()`: Take a screenshot of the current page.

Usage:

```
SeleniumSession$screenshot(timeout = 20)
```

Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: The base64-encoded PNG screenshot, as a string.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$screenshot()

session$close()
}
```

Method `print_page()`: Render the current page as a PDF.

Usage:
```
SeleniumSession$print_page(
  orientation = c("portrait", "landscape"),
  scale = 1,
  background = FALSE,
  width = NULL,
  height = NULL,
  margin = NULL,
  footer = NULL,
  header = NULL,
  shrink_to_fit = NULL,
  page_ranges = NULL,
  request_body = NULL,
  timeout = 20
)
```

Arguments:

`orientation` The page orientation, either "portrait" or "landscape".

`scale` The page scale, a number between 0.1 and 2.

`background` Whether to print the background of the page.

`width` The page width, in inches.

`height` The page height, in inches.

`margin` The page margin, in inches. Either a number, in which case the margin on all sides are set to that value, or a list of four numbers, with names `left`, `right`, `top`, and `bottom`, in which case the margin on each side is set individually.

`footer` The page footer, as a string.

`header` The page header, as a string.

`shrink_to_fit` Whether to shrink the page to fit the width and height.

`page_ranges` A list of page ranges (e.g. "1", "1-3") to print.

`request_body` A list of request body parameters to pass to the Selenium server, overriding the default body of the web request

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: The base64-encoded PDF, as a string.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$print_page()

session$close()
}
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
SeleniumSession$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SeleniumSession$new`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new(verbose = TRUE)

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$create_webelement`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

element <- session$find_element(using = "css selector", value = "*")

element2 <- session$create_webelement(id = element$id)

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$create_shadowroot`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

shadow_root <- session$create_shadowroot(id = "foo")

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$close`
## ------------------------------------------------

## Not run:
```

```
session <- SeleniumSession$new()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$status`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$status()

session$close()

session$status()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$get_timeouts`
## ------------------------------------------------

## Not run:

session <- SeleniumSession$new()

session$get_timeouts()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$set_timeouts`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$set_timeouts(script = 100)

session$get_timeouts()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$navigate`
## ------------------------------------------------
```

```
## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$close()

## End(Not run)

## ----------------------------------------------
## Method `SeleniumSession$current_url`
## ----------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$current_url()

session$close()

## End(Not run)

## ----------------------------------------------
## Method `SeleniumSession$back`
## ----------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$navigate("https://www.tidyverse.org")

session$back()

session$current_url()

session$close()

## End(Not run)

## ----------------------------------------------
## Method `SeleniumSession$forward`
## ----------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")
```

```
session$navigate("https://www.tidyverse.org")

session$back()

session$forward()

session$current_url()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$refresh`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$refresh()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$title`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$title()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$window_handle`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$window_handle()

session$close()
```

```
## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$close_window`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$new_window()

session$close_window()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$switch_to_window`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

handle <- session$window_handle()

handle2 <- session$new_window()$handle

session$switch_to_window(handle)

session$switch_to_window(handle2)

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$window_handles`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

handles <- session$window_handles()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$new_window`
## ------------------------------------------------
```

```
## Not run:
session <- SeleniumSession$new()

handle <- session$new_window()$handle

session$switch_to_window(handle)

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$switch_to_frame`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$switch_to_frame()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$switch_to_parent_frame`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$switch_to_frame()

session$switch_to_parent_frame()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$get_window_rect`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$get_window_rect()

session$close()
```

```
## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$set_window_rect`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$set_window_rect(width = 800, height = 600, x = 2, y = 3)

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$maximize_window`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$maximize_window()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$minimize_window`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$minimize_window()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$fullscreen_window`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$fullscreen_window()
```

```
session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$get_active_element`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$get_active_element()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$find_element`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "#download")

session$find_element(using = "xpath", value = "//div[contains(@class, 'col-xs')]/h1")

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$find_elements`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_elements(using = "css selector", value = "h1")

session$find_elements(using = "xpath", value = "//h1")

session$close()

## End(Not run)
```

```
## ------------------------------------------------
## Method `SeleniumSession$get_page_source`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$get_page_source()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$execute_script`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$execute_script("return 1")

session$execute_script("return arguments[0] + arguments[1]", 1, 2)

element <- session$find_element(value = "*")

session$execute_script("return arguments[0]", element)

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$execute_async_script`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$execute_async_script("
  let callback = arguments[arguments.length - 1];
  callback(1)
")

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$get_cookies`
## ------------------------------------------------
```

```
## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$get_cookies()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$get_cookie`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$add_cookie(list(name = "foo", value = "bar"))

session$get_cookie("foo")

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$add_cookie`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$add_cookie(list(name = "my_cookie", value = "1"))

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$delete_cookie`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")
```

```
session$add_cookie(list(name = "foo", value = "bar"))

session$delete_cookie("foo")

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$delete_all_cookies`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$delete_all_cookies()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$perform_actions`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

actions <- actions_stream(
  actions_press(keys$enter),
  actions_pause(0.5),
  actions_release(keys$enter)
)

session$perform_actions(actions)

session$close()

## End(Not run)

## ------------------------------------------------
## Method `SeleniumSession$release_actions`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")
```

```
actions <- actions_stream(
  actions_press("a")
)

session$perform_actions(actions, release_actions = FALSE)

session$release_actions()

session$close()

## End(Not run)

## ----------------------------------------------
## Method `SeleniumSession$dismiss_alert`
## ----------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$execute_script("alert('hello')")

session$dismiss_alert()

session$close()

## End(Not run)

## ----------------------------------------------
## Method `SeleniumSession$accept_alert`
## ----------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$execute_script("alert('hello')")

session$accept_alert()

session$close()

## End(Not run)

## ----------------------------------------------
## Method `SeleniumSession$get_alert_text`
## ----------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$execute_script("alert('hello')")

session$get_alert_text()
```

```
session$close()

## End(Not run)

## ---------------------------------------------
## Method `SeleniumSession$send_alert_text`
## ---------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$execute_script("prompt('Enter text:')")

session$send_alert_text("hello")

session$close()

## End(Not run)

## ---------------------------------------------
## Method `SeleniumSession$screenshot`
## ---------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$screenshot()

session$close()

## End(Not run)

## ---------------------------------------------
## Method `SeleniumSession$print_page`
## ---------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$print_page()

session$close()

## End(Not run)
```

---

selenium_server   *Download and start the Selenium server.*

Description

[Experimental]

Downloads the latest release of Selenium Server, and then runs it as a background process. You must have Java installed for this command to work.

Usage

```
selenium_server(
  version = "latest",
  selenium_manager = TRUE,
  interactive = TRUE,
  verbose = TRUE,
  temp = TRUE,
  path = NULL,
  echo_cmd = FALSE,
  extra_args = c()
)
```

Arguments

version	The version of Selenium Server to download and run. By default, the latest major or minor release is used.
selenium_manager	
	Whether to enable Selenium Manager, which will automatically download any missing drivers. Defaults to TRUE.
interactive	By default, if you don't have a version downloaded, you will be prompted to confirm that you want to download it, and the function will error if [rlang::is_interactive()](#) returns FALSE. To allow this function to work in a non-interactive setting, set this to FALSE.
verbose	Passed into [utils::download.file()](#). Note that setting this to FALSE will *not* disable the prompt if a file needs to be downloaded.
temp	Whether to use a temporary directory to download the Selenium Server .jar file. This will ensure that the file is deleted after it is used, but means that you will have to redownload the file with every new R session. If FALSE, the file is saved in your user data directory.
path	The path where the downloaded Selenium Server .jar file will be saved. Overrides temp.
echo_cmd	Passed into [processx::process$new()](#).
extra_args	A character vector of extra arguments to pass into the Selenium Server call.

Value

A [processx::process](#) object. Call <process>$kill() to stop the server.

See Also

The [package website](#) for more ways to start the Selenium server.

Examples

```
server <- selenium_server(interactive = FALSE)

server$kill()
```

---

selenium_server_available

Is a selenium server instance running?

Description

`selenium_server_available()` returns `TRUE` if a Selenium server is running on a given port and host. `wait_for_selenium_available()` waits for the Selenium server to become available for a given time.

`get_server_status()`, when given a port and host, figures out whether a Selenium server instance is running, and if so, returns its status. This is used by `selenium_server_available()` to figure out if the server is running.

Usage

```
selenium_server_available(
  port = 4444L,
  host = "localhost",
  verbose = FALSE,
  timeout = 20
)

wait_for_selenium_available(
  max_time = 60,
  port = 4444L,
  host = "localhost",
  verbose = FALSE,
  timeout = 20,
  error = FALSE
)

get_server_status(
  port = 4444L,
  host = "localhost",
  verbose = FALSE,
  timeout = 20
)
```

Arguments

port	The port that the Selenium server is using, so we can connect to it.
host	The host that the Selenium server is running on. This is usually 'localhost' (i.e. Your own machine).
verbose	Whether to print information about the web request that is sent.
timeout	How long to wait for a request to recieve a response before throwing an error.
max_time	The amount of time to wait for the Selenium server to become available.
error	Whether to throw an error if the web request throws an error after the timeout is exceeded. By default, a logical value is always returned.

Value

selenium_server_available() returns TRUE if a Selenium server is running, and FALSE otherwise, and wait_for_selenium_available() returns the same thing.

get_server_status() returns a list that can (but may not always) contain the following fields:

- ready: Whether the server is ready to be connected to. This should always be returned by the server.
- message: A message about the status of the server.
- uptime: How long the server has been running.
- nodes: Information about the slots that the server can take.

Examples

```
## Not run:
get_server_status()

selenium_server_available()

wait_for_selenium_available()

## End(Not run)
```

ShadowRoot   *Create a shadow root*

Description

A shadow DOM is a self-contained DOM tree, contained within another DOM tree. A shadow root is an element that contains a DOM subtree. This class represents a shadow root object, allowing you to select elements within the shadow root.

Public fields

id The id of the shadow root.

Methods

Public methods:

- ShadowRoot$new()
- ShadowRoot$find_element()
- ShadowRoot$find_elements()
- ShadowRoot$toJSON()
- ShadowRoot$clone()

Method new()**:** Initialize a new ShadowRoot object. This should not be called manually: instead use WebElement$shadow_root(), or SeleniumSession$create_shadow_root().

Usage:

```
ShadowRoot$new(session_id, req, verbose, id)
```

Arguments:

session_id The id of the session.

req, verbose Private fields of a SeleniumSession object.

id The id of the shadow root.

Returns: A ShadowRoot object.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

# Let's create our own Shadow Root using JavaScript
session$execute_script("
  const div = document.createElement('div');
  document.body.appendChild(div);
  div.attachShadow({mode: 'open'});
")

element <- session$find_element(using = "css selector", value = "div")

element$shadow_root()

session$close()
}
```

Method find_element()**:** Find an element in the shadow root.

Usage:

```
ShadowRoot$find_element(
 using = c("css selector", "xpath", "tag name", "link text", "partial link text"),
  value,
  request_body = NULL,
  timeout = 20
)
```

Arguments:

using The type of selector to use.

value The value of the selector: a string.

request_body A list of request body parameters to pass to the Selenium server, overriding the
   default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: A WebElement object.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

# Let's create our own Shadow Root using JavaScript
session$execute_script("
  const div = document.createElement('div');
  document.body.appendChild(div);
  const shadowRoot = div.attachShadow({mode: 'open'});
  const span = document.createElement('span');
  span.textContent = 'Hello';
  shadowRoot.appendChild(span);
")

element <- session$find_element(using = "css selector", value = "div")

shadow_root <- element$shadow_root()

shadow_root$find_element(using = "css selector", value = "span")

session$close()
}
```

Method `find_elements()`: Find all elements in a shadow root matching a selector.

Usage:

```
ShadowRoot$find_elements(
 using = c("css selector", "xpath", "tag name", "link text", "partial link text"),
 value,
 request_body = NULL,
 timeout = 20
)
```

Arguments:

using The type of selector to use.

value The value of the selector: a string.

request_body A list of request body parameters to pass to the Selenium server, overriding the
   default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: A list of WebElement objects.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

# Let's create our own Shadow Root using JavaScript
session$execute_script("
  const div = document.createElement('div');
  document.body.appendChild(div);
  const shadowRoot = div.attachShadow({mode: 'open'});
  const span = document.createElement('span');
  span.textContent = 'Hello';
  shadowRoot.appendChild(span);
  const p = document.createElement('p');
  p.textContent = 'Me too!';
  shadowRoot.appendChild(p);
")

element <- session$find_element(using = "css selector", value = "div")

shadow_root <- element$shadow_root()

shadow_root$find_elements(using = "css selector", value = "*")

session$close()
}
```

Method toJSON(): Convert an element to JSON. This is used by [SeleniumSession$execute_script()](#).

Usage:

```
ShadowRoot$toJSON()
```

Returns: A list, which can then be converted to JSON using `jsonlite::toJSON()`.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

# Let's create our own Shadow Root using JavaScript
session$execute_script("
  const div = document.createElement('div');
  document.body.appendChild(div);
  div.attachShadow({mode: 'open'});
")

element <- session$find_element(using = "css selector", value = "div")

shadow_root <- element$shadow_root()

result <- shadow_root$toJSON()

result
```

```
    jsonlite::toJSON(result, auto_unbox = TRUE)

    session$close()
    }
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ShadowRoot$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ShadowRoot$new`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

# Let's create our own Shadow Root using JavaScript
session$execute_script("
  const div = document.createElement('div');
  document.body.appendChild(div);
  div.attachShadow({mode: 'open'});
")

element <- session$find_element(using = "css selector", value = "div")

element$shadow_root()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `ShadowRoot$find_element`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

# Let's create our own Shadow Root using JavaScript
session$execute_script("
  const div = document.createElement('div');
  document.body.appendChild(div);
  const shadowRoot = div.attachShadow({mode: 'open'});
  const span = document.createElement('span');
  span.textContent = 'Hello';
```

```
    shadowRoot.appendChild(span);
")

element <- session$find_element(using = "css selector", value = "div")

shadow_root <- element$shadow_root()

shadow_root$find_element(using = "css selector", value = "span")

session$close()

## End(Not run)

## ------------------------------------------------
## Method `ShadowRoot$find_elements`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

# Let's create our own Shadow Root using JavaScript
session$execute_script("
  const div = document.createElement('div');
  document.body.appendChild(div);
  const shadowRoot = div.attachShadow({mode: 'open'});
  const span = document.createElement('span');
  span.textContent = 'Hello';
  shadowRoot.appendChild(span);
  const p = document.createElement('p');
  p.textContent = 'Me too!';
  shadowRoot.appendChild(p);
")

element <- session$find_element(using = "css selector", value = "div")

shadow_root <- element$shadow_root()

shadow_root$find_elements(using = "css selector", value = "*")

session$close()

## End(Not run)

## ------------------------------------------------
## Method `ShadowRoot$toJSON`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

# Let's create our own Shadow Root using JavaScript
session$execute_script("
  const div = document.createElement('div');
```

```
    document.body.appendChild(div);
    div.attachShadow({mode: 'open'});
")

element <- session$find_element(using = "css selector", value = "div")

shadow_root <- element$shadow_root()

result <- shadow_root$toJSON()

result

jsonlite::toJSON(result, auto_unbox = TRUE)

session$close()

## End(Not run)
```

WebElement   *Create a live element*

Description

This class represents a single element on the page. It is created using an existing [SeleniumSession](#) instance.

Public fields

id The id of the element, used to uniquely identify it on the page.

Methods

Public methods:

- [WebElement$new()](#)
- [WebElement$shadow_root()](#)
- [WebElement$find_element()](#)
- [WebElement$find_elements()](#)
- [WebElement$is_selected()](#)
- [WebElement$get_attribute()](#)
- [WebElement$get_property()](#)
- [WebElement$get_css_value()](#)
- [WebElement$get_text()](#)
- [WebElement$get_tag_name()](#)
- [WebElement$get_rect()](#)
- [WebElement$is_enabled()](#)
- [WebElement$computed_role()](#)

- `WebElement$computed_label()`
- `WebElement$click()`
- `WebElement$clear()`
- `WebElement$send_keys()`
- `WebElement$screenshot()`
- `WebElement$is_displayed()`
- `WebElement$toJSON()`
- `WebElement$clone()`

Method `new()`: Initialize a `WebElement` object. This should not be called manually: instead use SeleniumSession$create_webelement() if you have an element id. To find elements on the page, use SeleniumSession$find_element() and SeleniumSession$find_elements().

Usage:
```
WebElement$new(session_id, req, verbose, id)
```

Arguments:

`session_id` The id of the session that the element belongs to.

`req, verbose` Private fields of a SeleniumSession object.

`id` The element id.

Returns: A `WebElement` object.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

element <- session$find_element(using = "css selector", value = "#download")

session$close()
}
```

Method `shadow_root()`: A shadow DOM is a self-contained DOM tree, contained within another DOM tree. A shadow root is an element that contains a DOM subtree. This method gets the shadow root property of an element.

Usage:
```
WebElement$shadow_root(timeout = 20)
```

Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: A ShadowRoot object.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

# Let's create our own Shadow Root using JavaScript
```

```
session$execute_script("
  const div = document.createElement('div');
  document.body.appendChild(div);
  div.attachShadow({mode: 'open'});
")

element <- session$find_element(using = "css selector", value = "div")

shadow_root <- element$shadow_root()

session$close()
}
```

Method find_element(): Find the first element matching a selector, relative to the current element.

Usage:
```
WebElement$find_element(
  using = c("css selector", "xpath", "tag name", "link text", "partial link text"),
  value,
  request_body = NULL,
  timeout = 20
)
```

Arguments:

using The type of selector to use.

value The value of the selector: a string.

request_body A list of request body parameters to pass to the Selenium server, overriding the default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: A WebElement object.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

row <- session$find_element(using = "css selector", value = ".row")

logo_container <- row$find_element(using = "css selector", value = "p")

logo <- logo_container$find_element(using = "css selector", value = "img")

session$close()
}
```

Method find_elements(): Find all elements matching a selector, relative to the current element.

Usage:
```
WebElement$find_elements(
  using = c("css selector", "xpath", "tag name", "link text", "partial link text"),
  value,
  request_body = NULL,
  timeout = 20
)
```

Arguments:

using The type of selector to use.

value The value of the selector: a string.

request_body A list of request body parameters to pass to the Selenium server, overriding the default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: A list of WebElement objects.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

row <- session$find_element(using = "css selector", value = ".row")

links <- row$find_elements(using = "css selector", value = "a")

session$close()
}
```

Method `is_selected()`: Check if an element is currently selected.

Usage:
```
WebElement$is_selected(timeout = 20)
```

Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: A boolean value: TRUE or FALSE.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "#download")$is_selected()

session$close()
}
```

Method `get_attribute()`: Get an attribute from an element.

Usage:

```
WebElement$get_attribute(name, request_body = NULL, timeout = 20)
```

Arguments:

name The name of the attribute.

request_body A list of request body parameters to pass to the Selenium server, overriding the
        default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The value of the attribute: a string.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "a")$get_attribute("href")

session$close()
}
```

Method `get_property()`: Get a property from an element. Properties are similar to attributes,
but represent the HTML source code of the page, rather than the current state of the DOM.

Usage:

```
WebElement$get_property(name, request_body = NULL, timeout = 20)
```

Arguments:

name The name of the property.

request_body A list of request body parameters to pass to the Selenium server, overriding the
        default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The value of the property: a string.

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "a")$get_property("href")

session$close()
}
```

Method `get_css_value()`: Get the computed value of a CSS property.

Usage:

```
WebElement$get_css_value(name, request_body = NULL, timeout = 20)
```

Arguments:

name The name of the CSS property.

request_body A list of request body parameters to pass to the Selenium server, overriding the
   default body of the web request

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The value of the CSS property: a string.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "a")$get_css_value("color")

session$close()
}
```

Method get_text(): Get the text content of an element.

Usage:
```
WebElement$get_text(timeout = 20)
```

Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The text content of the element: a string.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "#download")$get_text()

session$close()
}
```

Method get_tag_name(): Get the tag name of an element.

Usage:
```
WebElement$get_tag_name(timeout = 20)
```

Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The tag name of the element: a string.

Examples:
```
\dontrun{
session <- SeleniumSession$new()
```

```
session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "#download")$get_tag_name()

session$close()
}
```

Method `get_rect()`: Get the dimensions and coordinates of an element.

Usage:
```
WebElement$get_rect(timeout = 20)
```

Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: A list containing the following elements:
- x: The x-coordinate of the element.
- y: The y-coordinate of the element.
- width: The width of the element in pixels.
- height: The height of the element in pixels.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "#download")$get_rect()

session$close()
}
```

Method `is_enabled()`: Check if an element is currently enabled.

Usage:
```
WebElement$is_enabled(timeout = 20)
```

Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: A boolean value: TRUE or FALSE.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "a")$is_enabled()

session$close()
}
```

Method `computed_role()`: Get the computed role of an element. The role of an element is usually "generic", but is often used when an elements tag name differs from its purpose. For example, a link that is "button-like" in nature may have a "button" role.

Usage:
```
WebElement$computed_role(timeout = 20)
```

Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: A string.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "a")$computed_role()

session$close()
}
```

Method `computed_label()`: Get the computed label of an element (i.e. The text of the label element that points to the current element).

Usage:
```
WebElement$computed_label(timeout = 20)
```

Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: A string.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "a")$computed_label()

session$close()
}
```

Method `click()`: Click on an element.

Usage:
```
WebElement$click(timeout = 20)
```

Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: The element, invisibly.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "a")$click()

session$close()
}
```

Method `clear()`: Clear the contents of a text input element.

Usage:
```
WebElement$clear(timeout = 20)
```
Arguments:

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: The element, invisibly.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.google.com")

session$find_element(using = "css selector", value = "textarea")$clear()

session$close()
}
```

Method `send_keys()`: Send keys to an element.

Usage:
```
WebElement$send_keys(..., request_body = NULL, timeout = 20)
```
Arguments:

`...` The keys to send (strings). Use [keys] for special keys, and use [key_chord()] to send keys combinations.

`request_body` A list of request body parameters to pass to the Selenium server, overriding the default body of the web request

`timeout` How long to wait for a request to recieve a response before throwing an error.

Returns: The element, invisibly.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.google.com")
```

```
input <- session$find_element(using = "css selector", value = "textarea")

input$send_keys("Hello")

input$send_keys(key_chord(keys$control, "a"), key_chord(keys$control, "c"))

input$send_keys(keys$control, "v")

input$get_attribute("value")

session$close()
}
```

Method screenshot(): Take a screenshot of an element.

Usage:
```
WebElement$screenshot(timeout = 20)
```
Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: The base64-encoded PNG screenshot, as a string.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "a")$screenshot()

session$close()
}
```

Method is_displayed(): Check if an element is displayed. This function may not work on all platforms.

Usage:
```
WebElement$is_displayed(timeout = 20)
```
Arguments:

timeout How long to wait for a request to recieve a response before throwing an error.

Returns: A boolean.

Examples:
```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "a")$is_displayed()

session$close()
}
```

Method toJSON(): Convert an element to JSON. This is used by SeleniumSession$execute_script().

Usage:

WebElement$toJSON()

Returns: A list, which can then be converted to JSON using jsonlite::toJSON().

Examples:

```
\dontrun{
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

result <- session$find_element(using = "css selector", value = "a")$toJSON()

result

jsonlite::toJSON(result, auto_unbox = TRUE)

session$close()
}
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

WebElement$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `WebElement$new`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

element <- session$find_element(using = "css selector", value = "#download")

session$close()

## End(Not run)

## ------------------------------------------------
## Method `WebElement$shadow_root`
## ------------------------------------------------

## Not run:
```

```
session <- SeleniumSession$new()

# Let's create our own Shadow Root using JavaScript
session$execute_script("
  const div = document.createElement('div');
  document.body.appendChild(div);
  div.attachShadow({mode: 'open'});
")

element <- session$find_element(using = "css selector", value = "div")

shadow_root <- element$shadow_root()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `WebElement$find_element`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

row <- session$find_element(using = "css selector", value = ".row")

logo_container <- row$find_element(using = "css selector", value = "p")

logo <- logo_container$find_element(using = "css selector", value = "img")

session$close()

## End(Not run)

## ------------------------------------------------
## Method `WebElement$find_elements`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

row <- session$find_element(using = "css selector", value = ".row")

links <- row$find_elements(using = "css selector", value = "a")

session$close()

## End(Not run)
```

```
## ----------------------------------------------
## Method `WebElement$is_selected`
## ----------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "#download")$is_selected()

session$close()

## End(Not run)

## ----------------------------------------------
## Method `WebElement$get_attribute`
## ----------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "a")$get_attribute("href")

session$close()

## End(Not run)

## ----------------------------------------------
## Method `WebElement$get_property`
## ----------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "a")$get_property("href")

session$close()

## End(Not run)

## ----------------------------------------------
## Method `WebElement$get_css_value`
## ----------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")
```

```
session$find_element(using = "css selector", value = "a")$get_css_value("color")

session$close()

## End(Not run)

## ------------------------------------------------
## Method `WebElement$get_text`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "#download")$get_text()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `WebElement$get_tag_name`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "#download")$get_tag_name()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `WebElement$get_rect`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "#download")$get_rect()

session$close()

## End(Not run)

## ------------------------------------------------
```

```
## Method `WebElement$is_enabled`
## ----------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "a")$is_enabled()

session$close()

## End(Not run)

## ----------------------------------------------
## Method `WebElement$computed_role`
## ----------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "a")$computed_role()

session$close()

## End(Not run)

## ----------------------------------------------
## Method `WebElement$computed_label`
## ----------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "a")$computed_label()

session$close()

## End(Not run)

## ----------------------------------------------
## Method `WebElement$click`
## ----------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")
```

```
session$find_element(using = "css selector", value = "a")$click()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `WebElement$clear`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.google.com")

session$find_element(using = "css selector", value = "textarea")$clear()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `WebElement$send_keys`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.google.com")

input <- session$find_element(using = "css selector", value = "textarea")

input$send_keys("Hello")

input$send_keys(key_chord(keys$control, "a"), key_chord(keys$control, "c"))

input$send_keys(keys$control, "v")

input$get_attribute("value")

session$close()

## End(Not run)

## ------------------------------------------------
## Method `WebElement$screenshot`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")
```

```
session$find_element(using = "css selector", value = "a")$screenshot()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `WebElement$is_displayed`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

session$find_element(using = "css selector", value = "a")$is_displayed()

session$close()

## End(Not run)

## ------------------------------------------------
## Method `WebElement$toJSON`
## ------------------------------------------------

## Not run:
session <- SeleniumSession$new()

session$navigate("https://www.r-project.org")

result <- session$find_element(using = "css selector", value = "a")$toJSON()

result

jsonlite::toJSON(result, auto_unbox = TRUE)

session$close()

## End(Not run)
```

Index