# Table of Contents

# Local Configuration

## Chrome

1. Download the latest ChromeDriver binary from here
2. Add it to your path, or tell Selenium where to find it
3. Create an instance of Chrome

```
require 'selenium-webdriver'
Selenium::WebDriver::Chrome::Service.executable_path = './chromedriver'
driver = Selenium::WebDriver.for :chrome
```

For more info:

- the Selenium wiki page for ChromeDriver
- the official user documentation

## Firefox

Available out of the box.

```
require 'selenium-webdriver'
driver = Selenium::WebDriver.for :firefox
```

For more info:

- the Selenium wiki page for FirefoxDriver

## Internet Explorer

Only available on Microsoft Windows.

1. Download the latest IEDriverServer from here
2. Add the downloaded file location to your path
3. Create an instance of Internet Explorer

```
require 'selenium-webdriver'
driver = Selenium::WebDriver.for :internet_explorer
```

For more info:

- the Selenium wiki page for InternetExplorerDriver

## Opera

Only works for version 12.16 or earlier. For newer versions of Opera, test using Chrome (since it uses the same back-end).

1. Download the latest Selenium Standalone Server from here
2. Create an environment variable pointing to the server file
3. Create an instance of Opera

```
require 'selenium-webdriver'
ENV['SELENIUM_SERVER_JAR'] = './selenium-server-standalone.jar'
driver = Selenium::WebDriver.for :opera
driver.get 'http://www.google.com'
driver.quit
```

For more info:

- The Selenium wiki page for OperaDriver

## Safari

Available out of the box as of version 2.21 of Selenium.

```
require 'selenium-webdriver'
driver = Selenium::WebDriver.for :firefox
```

For more info:

- the Selenium wiki page for SafariDriver

# Common Actions

## Find an element

```
# just one, the first one
driver.find_element(locator)

# all instances of the element
driver.find_elements(locator)
# returns an Array
```

## Work with a found element

```
# Chain actions together
driver.find_element(locator).click

# Store the element
element = driver.find_element(locator)
element.click
```

## Work with a collection of found elements

```
collection = driver.find_elements(locator)

# by name
collection.first.click
collection.last.click

# by index
collection[0].click
collection[-1].click
```

## Perform an action

```
element.click
element.submit      # submits a form
element.clear       # clearing an input field of its text
element.send_keys   # typing into an input field
```

## Ask a question

```
element.displayed? # is it visible?
element.enabled?   # can it be selected?
element.selected?  # is it selected?
```

## Retrieve information

```
# by attribute name
element.attribute('href')

# directly from an element
element.text
```

For more info:

- the `attribute` documentation

# Cloud Configuration

## Sauce Labs

1. Store your Sauce Labs Username and Access Key in environment variables
2. Specify the browser and operating system you want through Selenium's Capabilities
3. Create an instance of Selenium using the Sauce Labs end-point, passing in the Capabilities

```
ENV['SAUCE_USERNAME'] = 'your username goes here'
ENV['SAUCE_ACCESS_KEY'] = 'your access key goes here'

capabilities = Selenium::WebDriver::Remote::Capabilities.firefox
capabilities.version = "23"
capabilities.platform = "Windows XP"
driver = Selenium::WebDriver.for(
  :remote,
  :url => "http://SAUCE_USERNAME:SAUCE_ACCEESS_KEY@ondemand.saucelabs.com:80/wd/hub",
  :desired_capabilities => capabilities)
```

For more info:

- Sauce Labs Available Platforms page

# Cookies

## Retrieve Individual Cookie

```
cookie = driver.manage.cookie_named 'CookieName'
```

## Add

```
driver.manage.add_cookie(name: 'key', value: 'value')
```

## Delete

```
# one cookie
driver.manage.delete_cookie('CookieName')

# all cookies
driver.manage.delete_all_cookies
# does not delete third-party cookies, just the ones for the domain Selenium is
visiting
```

# Dropdown Lists

1. Find the dropdown list
2. Pass it into the Selenium Select helper function
3. Select from the list by its text or value

```
require 'selenium-webdriver'

driver.get 'http://the-internet.herokuapp.com/dropdown'
dropdown = @driver.find_element(id: 'dropdown')
select_list = Selenium::WebDriver::Support::Select.new(dropdown)
select_list.select_by(:text, "Option 1")
# select_list.select_by(:value, "1")
```

# Exception Handling

1. Rescue the relevant exceptions in a helper method, returning `false` for each
2. Create a convenience method to see if an element is displayed

```
def rescue_exceptions
  begin
    yield
  rescue Selenium::WebDriver::Error::NoSuchElementError
    false
  rescue Selenium::WebDriver::Error::StaleElementReferenceError
    false
  end
end

def is_displayed?(locator)
  rescue_exceptions { driver.find_element(locator).displayed? }
end

is_displayed? locator
# will return false if the element is not displayed
# otherwise, it will return true
```

For more info:

- a full list of Selenium exceptions

# File Transfers

## Upload

1. Find the text field for the upload form
2. Send text to it
3. Submit the form

```
require 'selenium-webdriver'

driver.get 'http://the-internet.herokuapp.com/upload'
uploader = driver.find_element(id: 'file-upload')
uploader.send_keys 'path of file you want to upload'
uploader.submit
```

## Download

1. Get the download link from Selenium
2. Use a third-party HTTP library to perform a HEAD request
3. Query the headers to look at the content type and content length

```
require 'selenium-webdriver'
require 'rspec-expectations'
require 'rest-client'

driver.get 'http://the-internet.herokuapp.com/download'
link = driver.find_element(css: 'a').attribute('href')
response = RestClient.head link
response.headers[:content_type].should == 'image/jpeg'
response.headers[:content_length].to_i.should > 0
```

## Download Secure Files

1. Get the download link from Selenium
2. Pull the session cookie from Selenium
3. Use a third-party HTTP library to perform a HEAD request using the session cookie
4. Query the headers to look at the content type and content length

```ruby
require 'selenium-webdriver'
require 'rspec-expectations'
require 'rest-client'

driver.get 'http://admin:admin@the-internet.herokuapp.com/download_secure'
link = driver.find_element(css: 'a').attribute('href')
driver.manage.cookie_named 'rack.session'
response = RestClient.head link, cookie: "#{cookie[:name]}=#{cookie[:value]};"
response.headers[:content_type].should == 'application/pdf'
response.headers[:content_length].to_i.should > 0
```

# Frames

1. Switch into the frame
2. Perform an action

## Nested Frames

```
require 'selenium-webdriver'
require 'rspec-expectations'

driver.get 'http://the-internet.herokuapp.com/frames'
driver.switch_to.frame('frame-top')
driver.switch_to.frame('frame-middle')
driver.find_element(id: 'content').text.should =~ /MIDDLE/
```

## IFrames

```
require 'selenium-webdriver'
require 'rspec-expectations'

driver.get 'http://the-internet.herokuapp.com/tinymce'
driver.switch_to.frame('mce_0_ifr')
  editor = @driver.find_element(id: 'tinymce')
  before_text = editor.text
  editor.clear
  editor.send_keys 'Hello World!'
  after_text = editor.text
after_text.should_not == before_text


# Hovers

1. Find the element
2. Create an action with the Selenium Action builder
3. Pass in the found element when calling the `move_to` action
4. Perform the action

```ruby
element = driver.find_element(locator)
driver.action.move_to(element).perform
```

For more info:
```

- the Selenium Action Builder `move_to` documentation

# JavaScript

## Execution

```
driver.execute_script('your javascript goes here')
```

## Alerts

```
popup = driver.switch_to.alert
popup.accept
# or popup.dismiss
```

# Key Presses

```
driver.action.send_keys(key)
# e.g., driver.action.send_keys(:tab)
```

For more info:

- the Selenium Action Builder `send_keys` documentation
- a list of available keyboard keys and their trigger values

# Multiple Windows

## A simple way

```
driver.switch_to.window(driver.window_handles.first)
driver.switch_to.window(driver.window_handles.last)
```

Caveat: The order of the window handles is not consistent across all browsers. Some return in the order opened, others alphabetically.

## A solid way

```ruby
main_window = @driver.window_handle
# action that triggers a new window
windows = @driver.window_handles
windows.each do |window|
  if main_window != window
    @new_window = window
  end
end
```

# Screenshots

## Simple screenshot

```
driver.save_screenshot "screenshot.png"
```

## Uniquely named screenshot by timestamp

```
driver.save_screenshot "./#{Time.now.strftime("failshot__%d_%m_%Y__%H_%M_%S")}.png"
```

For more info:

- strftime reference and sandbox

# Waiting

## Implicit Wait

- Only needs to be configured once
- Tells Selenium to wait for a specified amount of time before raising a `NoSuchElementError` exception
- Can be overridden with an explicit wait

```
driver.manage.timeouts.implicit_wait = 3
```

For more info:

- Explicit vs Implicit Waits

## Explicit Waits

- Specify an amount of time and an action
- Selenium will try the action repeatedly until either:
  - the action can be accomplished
  - the amount of time has been reached (and throw a timeout exception)

```
wait = Selenium::WebDriver::Wait.new(timeout: seconds)
wait.until { driver.find_element(locator).displayed? }
```

For more info:

- Explicit vs Implicit Waits