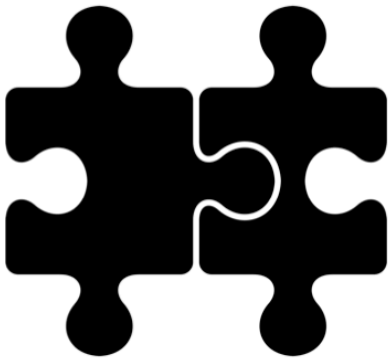
SELENIUM CHEAT SHEETS

Ruby Edition



by Dave Haeffner

Version 2.1.0

Table of Contents

Chapter 1

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

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

```
require 'selenium-webdriver'
geckodriver = File.join(Dir.pwd, 'vendor', 'geckodriver')
driver = Selenium::WebDriver.for :firefox, driver_path: geckodriver
```

To use the legacy FirefoxDriver:

1. Specify desired capabilities with `marionette` set to `false`
2. Pass in the desired capabilities when creating an instance of Firefox

```
require 'selenium-webdriver'
driver = Selenium::WebDriver.for :firefox, desired_capabilities: { marionette: false }
```

For more info:

- the Selenium wiki page for FirefoxDriver
- the geckodriver documentation from Mozilla

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

Safari

- Since Selenium 2.45.0, you need to manually install the SafariDriver browser extension
- Download it from here
- Double-click to install it
- Click `Trust` when prompted

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

For more info:

- SafariDriver on the Selenium Wiki
- Elemental Selenium tip 69

Chapter 2
Cloud Configuration

Sauce Labs

Initial Setup

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

Setting the Job Status

1. Install the `sauce_whisk` gem
2. Add `require 'sauce_whisk'` to your test harness configuration
3. Use `sauce_whisk` to mark the Sauce job as passed or failed by using Selenium's `session_id`

```ruby
# an RSpec example
require 'sauce_whisk'

after(:each) do |example|
  if example.exception.nil?
    SauceWhisk::Jobs.pass_job @driver.session_id
  else
    SauceWhisk::Jobs.fail_job @driver.session_id
  end
  @driver.quit
end
```

Using Sauce Connect for Private Apps

1. Download Sauce Connect
2. Start the Sauce Connect tunnel (e.g., `bin/sc -u YOUR_USERNAME -k YOUR_ACCESS_KEY`)
3. Run your tests

Chapter 3
Headless Configuration

GhostDriver via PhantomJS

1. Download PhantomJS
2. Start up PhantomJS with WebDriver support (specifying a port to run on)
3. Point your tests at PhantomJS
4. Run your tests

```
phantomjs --webdriver=8001
PhantomJS is launching GhostDriver...
[INFO  - 2014-04-16T02:07:51.015Z] GhostDriver - Main - running on port 8001
```

```
@driver = Selenium::WebDriver.for :remote, url: 'http://localhost:8001'
```

Xvfb

X virtual framebuffer (Xvfb) is an in-memory display server for UNIX-like operating systems (e.g., Linux).

Install

For Debian based systems

```
apt-get install xvfb firefox
```

For RedHat systems

```
yum install Xvfb firefox
```

Option 1

```
Xvfb :99 &
export DISPLAY=:99
rspec example_spec.rb
```

This approach will keep Xvfb running in the background until it is manually stopped.

Option 2

```
xvfb-run rspec example_spec.rb
```

This will start Xvfb and stop it after the test run completes.

Option 3

Use the headless gem to start and stop Xvfb in your test code.

```ruby
require 'headless'

# before your browser setup
@headless = Headless.new
@headless.start

# after your browser teardown
@headless.destroy
```

For more info:

- Wikipedia write-up on Xvfb
- Tip 38 on Elemental Selenium

Chapter 4

Common Actions

Visit a page

```
driver.get 'url'
# e.g., driver.get 'http://the-internet.herokuapp.com'
```

Find an element

Works using locators, which are covered in the next section.

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

# all instances of the element on the page
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       # click on an element
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:

- Selenium's Element API documentation
- Selenium's `attribute` Element API documentation

Chapter 5
Locators

Guiding principles

Good Locators are:

- unique
- descriptive
- unlikely to change

Be sure to:

1. Start with ID and Class
2. Use CSS selectors (or XPath) when you need to traverse
3. Talk with a developer on your team when the app is hard to automate
    1. tell them what you're trying to automate
    2. work with them to get more semantic markup added to the page

ID

```
driver.find_element(id: 'username')
```

Class

```
driver.find_elements(class: 'dues')
```

CSS Selectors

```
# Example usage
driver.find_element(css: '#username')
driver.find_element(css: '.dues')
```

Approach	Locator	Description
ID	`#example`	`#` denotes an ID
Class	`.example`	`.` denotes a Class
Classes	`.flash.success`	use `.` in front of each class for multiple
Direct child	`div > a`	finds the element in the next child
Child/subschild	`div a`	finds the element in a child or child's child
Next sibling	`input.username + input`	finds the next adjacent element
Attribute values	`form input[name='username']`	a great alternative to id and class matches
Attribute values	`input[name='continue'][type='button']`	can chain multiple attribute filters together
Location	`li:nth-child(4)`	finds the 4th element only if it is an li
Location	`li:nth-of-type(4)`	finds the 4th li in a list
Location	`*:nth-child(4)`	finds the 4th element regardless of type
Sub-string	`a[id^='beginning_']`	finds a match that starts with (prefix)
Sub-string	`a[id$='_end']`	finds a match that ends with (suffix)
Sub-string	`a[id*='gooey_center']`	finds a match that contains (substring)
Inner text	`a:contains('Log Out')`	an alternative to substring matching

NOTE: Older browser (e.g., Internet Explorer 8) don't support CSS Pseudo-classes, so some of these locator approaches won't work (e.g., Location matches and Inner text matches).

For more info:

- CSS vs XPath benchmarks
- CSS and XPath Examples
- CSS selector game
- The difference between nth-child and nth-of-type
- w3schools CSS Selectors Reference
- w3schools XPath Syntax Reference
- How to verify your locators

Chapter 6

Cookies

Retrieve Individual Cookie

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

Add

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

Delete

```
# Delete a single cookie
driver.manage.delete_cookie('CookieName')

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

Chapter 7

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")
```

Chapter 8
Exception Handling

1. Rescue the relevant exceptions, returning `false` for each
2. Place in a helper method for easy reuse

```
def is_displayed?(locator)
  begin
    driver.find_element(locator).displayed?
  rescue Selenium::WebDriver::Error::NoSuchElementError
    false
  rescue Selenium::WebDriver::Error::StaleElementReferenceError
    false
  end
end

is_displayed? locator
# Returns false if the element is not displayed.
# Otherwise, returns true.
```

For more info:

- Selenium's list of exceptions

Chapter 9
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 with Firefox

1. Create a Selenium profile configuration object
2. Specify the download method that enables specifying a path
3. Specify a folder path to download files to
4. Specify the MIME type of the files you want to download
5. Disable the Firefox PDF viewer (if downloading PDF files)
6. Load the profile configuration object when creating an instance of Selenium

```
profile = Selenium::WebDriver::Firefox::Profile.new
profile['browser.download.folderList'] = 2
profile['browser.download.dir'] = 'path to download dir'
profile['browser.helperApps.neverAsk.saveToDisk'] = 'image/jpeg, application/pdf'
profile['pdfjs.disabled'] = true

@driver = Selenium::WebDriver.for :firefox, profile: profile
```

For more info:

- A list of MIME types
- A list of all of Firefox's available preferences
- Tip 2 on Elemental Selenium

Download with an HTTP Library

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
expect(response.headers[:content_type]).to eql 'image/jpeg'
expect(response.headers[:content_length].to_i).to > 0
```

Download Secure Files with an HTTP Library

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

```
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]};"
expect(response.headers[:content_type]).to eql 'image/jpeg'
expect(response.headers[:content_length].to_i).to > 0
```

Chapter 10
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')
expect(driver.find_element(id: 'content').text).to eql 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
expect(after_text).not_to eql before_text
```

Chapter 11
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

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

For more info:

- Selenium's Action Builder `move_to` documentation

Chapter 12

JavaScript

Execution

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

Alerts

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

Chapter 13

Key Presses

```ruby
# find and send to an element
@driver.get 'http://the-internet.herokuapp.com/key_presses'
@driver.find_element(class: 'example').send_keys :space

# send to the current element in focus
@driver.action.send_keys(:tab).perform
```

For more info:

- Selenium's Action Builder `send_keys` documentation
- A list of available keyboard keys and their trigger values
- Tip 61 on Elemental Selenium

Chapter 14

Multiple Windows

A simple way

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

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

A browser agnostic way

```
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

@driver.get 'http://the-internet.herokuapp.com/windows'
main_window = @driver.window_handle
@driver.find_element(css: '.example a').click # triggers a new window
windows = @driver.window_handles
new_window = windows.select do |window|
  window != main_window
end
```

For more info:

- Tip 4 on Elemental Selenium

Chapter 15

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
- Tip 16 on Elemental Selenium

Chapter 16

Waiting

Implicit Wait

- Only needs to be configured once
- Tells Selenium to wait for a specified amount of time before raising a `NoSuchElementError` exception
- Less flexible than explicit waits

```
driver.manage.timeouts.implicit_wait = 5
```

Explicit Waits

- Recommended way to wait in your tests
- 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 TimeoutError)

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

For more info:

- Explicit vs Implicit Waits
- The case against using Implicit and Explicit Waits together