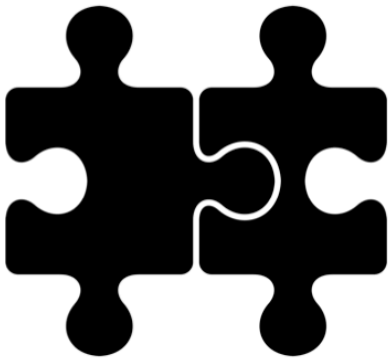
SELENIUM CHEAT SHEETS

JavaScript Edition



by Dave Haeffner

Version 4.0.0

Table of Contents

1. Local Configuration
2. Cloud Configuration
3. Common Commands
4. Locators
5. Exception Handling
6. Waiting

Chapter 1
Local Configuration

Firefox

Option 1

1. Download the latest geckodriver binary
2. Add its location to your path
3. Create an instance

```
const { Builder } = require('selenium-webdriver');
const path = require('path');
process.env.PATH += path.delimiter + path.join(__dirname, '..', 'vendor')
driver = new Builder().forBrowser('firefox').build();
```

Option 2

1. Use `npm` to install the latest geckodriver binary
2. Create an instance

```
> npm install geckodriver
```

```
driver = new Builder().forBrowser('firefox').build();
```

NOTE: For more information about geckodriver check out its project page

Chrome

Option 1

1. Download the latest ChromeDriver
2. Add its location to your path
3. Create an instance

```
const { Builder } = require('selenium-webdriver');
const path = require('path');
process.env.PATH += path.delimiter + path.join(__dirname, '..', 'vendor')
driver = new Builder().forBrowser('chrome').build();
```

Option 2

1. Use `npm` to install the latest geckodriver binary
2. Create an instance

```
> npm install chromedriver
```

```
driver = new Builder().forBrowser('chrome').build();
```

NOTE: For more information about ChromeDriver check out its project page

Internet Explorer

Option 1

1. Download the latest IEDriverServer.exe
2. Add its location to your path
3. Create an instance

```
const { Builder } = require('selenium-webdriver');
const path = require('path');
process.env.PATH += path.delimiter + path.join(__dirname, '..', 'vendor')
driver = new Builder().forBrowser('internet explorer').build();
```

Option 2

1. Use `npm` to install the latest IEDriverServer binary
2. Create an instance

```
> npm install iedriver
```

NOTE: There is additional setup required to make Internet Explorer work with Selenium. For more information check out the Selenium project Wiki page for InternetExplorerDriver.

Edge

In order to use Microsoft Edge you need to have access to Windows 10.

Option 1

1. Download EdgeDriver
2. Add its location to your path
3. Create an instance

```
const { Builder } = require('selenium-webdriver');
const path = require('path');
process.env.PATH += path.delimiter + path.join(__dirname, '..', 'vendor')
driver = new Builder().forBrowser('edge').build();
```

Option 2

1. Use `npm` to install the latest IEDriverServer binary
2. Create an instance

```
> npm install edgedriver
```

NOTE: You can download a free virtual machine with Windows 10 from Microsoft's Modern.IE developer portal. After that you need to download the appropriate **`Microsoft WebDriver`** server for your build of Windows. To find that go to **`Start`**, **`Settings`**, **`System`**, **`About`** and locate the number next to **`OS Build`** on the screen.

Safari

For Safari, you'll need SafariDriver, which ships with the latest version of Safari.

You just need to enable it from the command-line.

```
> safaridriver --enable
```

```
const { Builder } = require('selenium-webdriver');
driver = new Builder().forBrowser('safari').build();
```

NOTE: For additional details, or information on setup requirements for older versions of macOS, see the SafariDriver documentation from Apple.

Chapter 2
Cloud Configuration

Sauce Labs

Initial Setup

1. Create run-time flags with sensible defaults that can be overridden
2. Specify the browser and operating system you want through Desired Capabilitaies
3. Connect to Sauce Labs' end-point through the Desired Capabilities
4. Store the WebDriver instance returned for use in your tests

```
// filename: lib/config.js
module.exports = {
  host: process.env.HOST || 'saucelabs',
  sauce: {
    username: process.env.SAUCE_USERNAME,
    accessKey: process.env.SAUCE_ACCESS_KEY,
    platform: process.env.PLATFORM || 'Windows 7',
    browserName: process.env.BROWSER || 'internet explorer',
    version: process.env.BROWSER_VERSION || '11.0',
  },
};
```

```
// filename: lib/DriverFactory.js
let builder = new Builder();
const url = 'http://ondemand.saucelabs.com:80/wd/hub';
builder.usingServer(url);
builder.withCapabilities(this.config.sauce);
const driver = builder.build();
```

For more info:

- Sauce Labs Available Platforms page
- Sauce Labs Automated Test Configurator

Setting the Test Name

1. Grab the test class and test method name dynamically after the test runs (in the `afterEach`)
2. Update the Sauce Labs job through Selenium's JavaScript executor

```
const testName = this.currentTest.fullTitle();
driver.executeScript('sauce:job-name=' + testName);
```

Setting the Job Status

1. Grab the test result dynamically after the test runs (in the `afterEach`)
2. Update the Sauce Labs job through Selenium's JavaScript executor

```
const testResult = this.currentTest.state === 'passed'
driver.executeScript('sauce:job-result=' + testResult);
```

Chapter 3

Common Commands

Visit a page

```
await driver.get('http://the-internet.herokuapp.com');
```

Find an element

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

```
// find just one, the first one Selenium finds
await driver.findElement(locator);

// find all instances of the element on the page
// returns a collection
await driver.findElements(locator);
```

Work with a found element

```
// chain commands together
await driver.findElement(locator).click();

// store the element
// and then perform a command with it
const element = await driver.findElement(locator);
await element.click();
```

Perform an action

```
await element.click();                  // clicks an element
await element.submit();                 // submits a form
await element.clear();                  // clears an input field of its text
await element.sendKeys('input text');   // types into an input field
```

Ask a question

Each of these returns a Boolean.

```
await element.isDisplayed();    // is it visible?
await element.isEnabled();      // can it be selected?
await element.isSelected();     // is it selected?
```

Retrieve information

```
// by attribute name
await element.getAttribute('href');

// directly from an element
await element.getText();
```

For more info:

- the WebElement documentation for the Selenium JavaScript bindings

Chapter 4
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

```
await driver.findElement(By.id('username'));
```

Class

```
await driver.findElement(By.className('dues'));
```

CSS Selectors

```
await driver.findElement(By.css('#example'));
```

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

For more info see one of the following resources:

- CSS Selector Game
- CSS & XPath Examples by Sauce Labs
- The difference between nth-child and nth-of-type
- CSS vs. XPath Selenium benchmarks
- CSS Selectors Reference
- XPath Syntax Reference

Chapter 5

Exception Handling

1. Wrap the command that could throw an error inside of a `try` / `catch` block
2. In the `catch` block, `return` your preferred result

```
try {
  return await this.find(locator).isDisplayed()
} catch (error) {
  return false
}
```

For more info see:

- the MDN article on `try...catch`

Chapter 6

Waiting

Implicit Wait

- Specify a timeout during test setup (in milliseconds)
- For every command that Selenium is unable to complete, it will retry it until either:
  - the action can be accomplished, or
  - the amount of time specified has been reached and raise an exception (typically `NoSuchElementError`)
- Less flexible than explicit waits
- Not recommended

```
driver.manage().setTimeouts({ implicit: 15000 });
```

Explicit Waits

- Specify a timeout (in milliseconds) and an expected condition to wait for
- Selenium will check for the expected condition repeatedly until either:
  - is is successful, or
  - the amount of time specified has been reached and raise an exception
- Recommended way to wait in your tests

```
const Until = require('selenium-webdriver').until;
await this.driver.wait(Until.elementLocated(locator), timeout)
```

For more info:

- Explicit vs Implicit Waits
- Selenium JavaScript bindings documentation for explicit waits