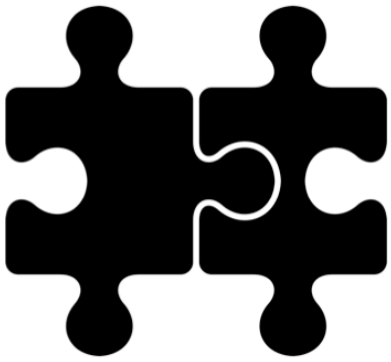
SELENIUM CHEAT SHEETS

C# Edition



by Dave Haeffner

Version 4.0.0

Table of Contents

Chapter 1

Local Configuration

Firefox

Option 1

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

```
using OpenQA.Selenium.Firefox;
IWebDriver = new FirefoxDriver();
```

Option 2

1. Download the latest geckodriver binary
2. Add its location to a system property in your setup code
3. Create an instance

```
using OpenQA.Selenium.Firefox;
var Service = FirefoxDriverService.CreateDefaultService(VendorDirectory);
IWebDriver = new FirefoxDriver(Service);
```

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

Chrome

Option 1

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

```
using OpenQA.Selenium.Chrome;
IWebDriver = new ChromeDriver();
```

Option 2

1. Download the latest ChromeDriver binary

2. Add its location to a system property in your setup code
3. Create an instance

```
using OpenQA.Selenium.Chrome;
var Service = ChromeDriverService.CreateDefaultService(VendorDirectory);
IWebDriver = new ChromeDriver(Service);
```

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

Internet Explorer

Option 1

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

```
using OpenQA.Selenium.IE;
IWebDriver = new InternetExplorerDriver();
```

Option 2

1. Download the latest IEDriverServer
2. Add its location to a system property in your setup code
3. Create an instance

```
using OpenQA.Selenium.IE;
var Service = InternetExplorerDriverService.CreateDefaultService(VendorDirectory);
IWebDriver = new InternetExplorerDriver(Service);
```

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

```
using OpenQA.Selenium.Edge;
IWebDriver = new EdgeDriver();
```

Option 2

1. Download EdgeDriver
2. Add its location to a system property in your setup code
3. Create an instance

```
using OpenQA.Selenium.Edge;
var Service = EdgeDriverService.CreateDefaultService(VendorDirectory);
IWebDriver = new EdgeDriver(Service);
```

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
```

```
using OpenQA.Selenium.Safari;
IWebDriver = new SafariDriver();
```

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. Store the browser/OS parameters
2. Pass them to a browser option object
3. Create an instance of `RemoteWebDriver` using Sauce Labs' end-point using the options object from #2
4. Store the instance in a field variable for use in your tests
5. Repeat for each browser you want to use on Sauce Labs

```
BrowserName     = System.Environment.GetEnvironmentVariable("BROWSER_NAME") ?? "ie";
BrowserVersion  = System.Environment.GetEnvironmentVariable("BROWSER_VERSION") ??
"10.0";
PlatformName    = System.Environment.GetEnvironmentVariable("PLATFORM_NAME") ??
"Windows 8";
var sauceUsername = System.Environment.GetEnvironmentVariable("SAUCE_USERNAME");
var sauceAccessKey = System.Environment.GetEnvironmentVariable("SAUCE_ACCESS_KEY");
var url = new Uri($
"http://{sauceUsername}:{sauceAccessKey}@ondemand.saucelabs.com:80/wd/hub");
ChromeOptions options = new ChromeOptions();
options.PlatformName = PlatformName;
options.BrowserVersion = BrowserVersion;
Driver = new RemoteWebDriver(url, options.ToCapabilities());
```

For more info see:

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

Setting the Test Name

1. Pull the test name out of NUnit's `TestContext`
2. Pass the name to Sauce Labs using the JavaScript executor

```
var testName = TestContext.CurrentContext.Test.Name;
((IJavaScriptExecutor)Driver).ExecuteScript("sauce:job-name=" + testName);
```

Setting the Job Status

1. Check the test result after the test completes
2. Use the JavaScript executor to pass the result onto Sauce Labs
3. BONUS POINTS: Output the Sauce Labs job URL to the console

```
bool testPassed = TestContext.CurrentContext.Result.Outcome.Status.ToString() ==
"Passed";
((IJavaScriptExecutor)Driver).ExecuteScript("sauce:job-result=" + (testPassed ?
"passed" : "failed"));
if(!testPassed)
{
    TestContext.WriteLine($"See a job at
https://saucelabs.com/tests/{((RemoteWebDriver)Driver).SessionId}");
}
```

Chapter 3

Common Actions

Visit a page

```
Driver.Navigate().GoToUrl("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
Driver.FindElement(locator);

// find all instances of the element on the page
Driver.FindElements(locator);
// returns a collection
```

Work with a found element

```
// chain actions together
Driver.FindElement(locator).Click();

// store the element
IWebElement Element = Driver.FindElement(locator);
Element.Click();
```

Perform an action

```
Element.Click();                  // clicks an element
Element.Clear();                  // clears an input field of it's text
Element.SendKeys("input text");   // types text into an input field
```

Ask a question

Each of these returns a Boolean.

```
Element.Displayed;    // is it visible to the human eye?
Element.Enabled;      // can it be selected?
Element.Selected;     // is it selected?
```

Retrieve information

Each of these returns a String.

```
// by attribute name
Element.GetAttribute("href");

// directly from an element
Element.Text;
```

For more info see:

- Selenium IWebElement API Documentation

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

```
Driver.FindElement(By.Id("username"));
```

Class

```
driver.findElement(By.ClassName("dues"));
```

CSS Selectors

```
Driver.FindElement(By.CssSelector("#username"));
Driver.FindElement(By.CssSelector(".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

For more info see:

- 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. Try the action you want
2. Catch the relevant exception and return `false` instead

```
try {
  return Find(locator).Displayed;
} catch(OpenQA.Selenium.NoSuchElementException) {
  return false;
}
```

For more info see:

- Selenium WebDriverException API Documentation

Chapter 6
Waiting

Implicit Wait

- Specify a timeout in milliseconds (typically during test setup)
- 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().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(10));
```

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

```
WebDriverWait Wait = new WebDriverWait(Driver, System.TimeSpan.FromSeconds(10));
Wait.Until(ExpectedConditions.ElementIsVisible(locator));
return true;
```

For more info see:

- Explicit vs. Implicit Waits