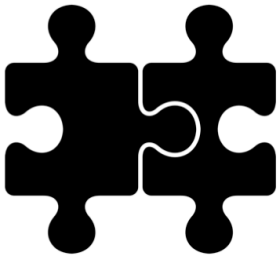
SELENIUM CHEAT SHEETS

Java 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

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

protected WebDriver driver;
driver = 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

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

protected WebDriver driver;
System.setProperty("webdriver.gecko.driver", "/path/to/geckodriver");
driver = new FirefoxDriver();
```

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

Chrome

Option 1

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

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

protected WebDriver driver;
driver = new ChromeDriver();
```

Option 2

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

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

protected WebDriver driver;
System.setProperty("webdriver.chrome.driver", "/path/to/chromedriver");
driver = new ChromeDriver();
```

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

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.ie.InternetExplorerDriver;

protected WebDriver driver;
driver = 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

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.ie.InternetExplorerDriver;

protected WebDriver driver;
System.setProperty("webdriver.ie.driver", "/path/to/iedriver");
driver = new InternetExplorerDriver();
```

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

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.edge.EdgeDriver;

protected WebDriver driver;
driver = new EdgeDriver();
```

Option 2

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

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.edge.EdgeDriver;

protected WebDriver driver;
System.setProperty("webdriver.edge.driver", "/path/to/edgedriver");
driver = new 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
```

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.safari.SafariDriver;

protected WebDriver driver;
driver = 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. Create field variables with sensible defaults that can be overridden at run-time
2. Specify the browser and operating system you want through Selenium's `DesiredCapabilities`
3. Create an instance of `RemoteWebDriver` using Sauce Labs' end-point -- providing your credentials and `DesiredCapabilities`
4. Store the instance in a field variable

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;

final String browserName    = System.getProperty("browserName", "firefox");
final String browserVersion = System.getProperty("browserVersion", "33");
final String platformName   = System.getProperty("platformName", "Windows XP");
final String sauceUser      = System.getenv("SAUCE_USERNAME");
final String sauceKey       = System.getenv("SAUCE_ACCESS_KEY");

DesiredCapabilities capabilities = new DesiredCapabilities();
capabilities.setCapability("browserName", browserName);
capabilities.setCapability("browserVersion", browserVersion);
capabilities.setCapability("platformName", platformName);

String sauceUrl = String.format("http://%s:%s@ondemand.saucelabs.com:80/wd/hub",
        sauceUser, sauceKey);

protected WebDriver driver;
driver = new RemoteWebDriver(new URL(sauceUrl), capabilities);
```

For more info see:

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

Setting the Test Name

1. Create a field variable to store the test name in
2. Add a Test Watcher Rule that uses the `starting()` method
3. Grab the display name of the test from within the Test Watcher and store it in the field variable
4. Pass the field variable value as a `"name"` capability in `DesiredCapabilities`

```
import org.junit.Rule;
import org.junit.rules.TestRule;
import org.junit.rules.TestWatcher;
import org.junit.runner.Description;

private String testName;

@Rule
public TestRule watcher = new TestWatcher() {
    protected void starting(Description description) {
        testName = description.getDisplayName();
    }
};

DesiredCapabilities capabilities = new DesiredCapabilities();
capabilities.setCapability("browserName", browserName);
capabilities.setCapability("browserVersion", browserVersion);
capabilities.setCapability("platformName", platformName);
capabilities.setCapability("name", testName);
```

For more info see:

- JUnit TestWatcher Rules documentation

Setting the Job Status

1. Install the `SauceREST` library
2. Create field variables to store `SauceREST` session and the Selenium session ID
3. Grab and store the Selenium session ID after a Sauce Labs instance is created
4. Add `failed()` and `succeeded()` Test Watcher methods
5. Create an instance of `SauceREST` to mark the Sauce job as passed or failed by using the Selenium session ID
6. BONUS POINTS: output the Sauce Labs job URL to the console when a test fails

```
<!-- filename: pom.xml -->
...
        <dependency>
            <groupId>com.saucelabs</groupId>
            <artifactId>saucerest</artifactId>
            <version>1.0.40</version>
            <scope>test</scope>
        </dependency>

    </dependencies>

</project>
```

```
// other import statements omitted for brevity
import com.saucelabs.saucerest.SauceREST;

protected WebDriver driver;
private String sessionId;
private SauceREST sauceClient;

driver = new RemoteWebDriver(new URL(sauceUrl), capabilities);
sessionId = ((RemoteWebDriver) driver).getSessionId().toString();
sauceClient = new SauceREST(sauceUser, sauceKey);

@Rule
public TestRule watcher = new TestWatcher() {

    @Override
    protected void failed(Throwable throwable, Description description) {
        if (host.equals("saucelabs")) {
            sauceClient.jobFailed(sessionId);
            System.out.println(String.format("https://saucelabs.com/tests/%s",
sessionId));
        }
    }

    @Override
    protected void succeeded(Description description) {
        if (host.equals("saucelabs")) {
            sauceClient.jobPassed(sessionId);
        }
    }

};
```

Chapter 3

Common Actions

Visit a page

```
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
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
WebElement 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.isDisplayed();    // is it visible to the human eye?
element.isEnabled();      // can it be selected?
element.isSelected();     // is it selected?
```

Retrieve information

Each of these returns a String.

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

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

For more info see:

- the Selenium WebElement 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

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

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 driver.findElement(locator).isDisplayed();
} catch (org.openqa.selenium.NoSuchElementException exception) {
    return false;
}
```

For more info see:

- the 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(5, TimeUnit.SECONDS);
```

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, timeout);
wait.until(condition);
// wait.until(ExpectedConditions.visibilityOfElementLocated(locator));
```

For more info see:

- Explicit vs. Implicit Waits
- Selenium documentation on explicit waits
- Selenium Java bindings documentation for ExpectedConditions