



Selenium WebDriver Recipes in Node.js

The Problem Solving Guide to Selenium WebDriver



Zhimin Zhan

Selenium WebDriver Recipes in Node.js

The problem solving guide to Selenium WebDriver in JavaScript

Zhimin Zhan

This book is for sale at <http://leanpub.com/selenium-webdriver-recipes-in-nodejs>

This version was published on 2021-05-01

ISBN 978-1537328256



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2016 - 2021 Zhimin Zhan

Also By **Zhimin Zhan**

[Watir Recipes](#)

[Selenium WebDriver Recipes in Ruby](#)

[Selenium WebDriver Recipes in Java](#)

[Learn Ruby Programming by Examples](#)

[Learn Swift Programming by Examples](#)

[Selenium WebDriver Recipes in Python](#)

[API Testing Recipes in Ruby](#)

[Practical Continuous Testing](#)

Contents

Preface	i
Who should read this book	ii
How to read this book	ii
Recipe test scripts	ii
Send me feedback	iii
1. Introduction	1
1.1 Selenium language bindings	1
1.2 Install Node.JS	3
1.3 Write first script with JavaScript editor	5
1.4 Install Selenium WebDriver	6
1.5 Run script	7
1.6 Cross browser testing	7
1.7 Mocha Test Framework	10
1.8 Create package.json file	13
1.9 Run recipe test scripts	14
2. Locating web elements	16
2.1 Start browser	16
2.2 Find element by ID	17
2.3 Find element by Name	18
2.4 Find element by Link Text	18
2.5 Find element by Partial Link Text	18
2.6 Find element by XPath	19
2.7 Find element by Tag Name	20
2.8 Find element by Class	21
2.9 Find element by CSS	21
2.10 Chain findElement to find child elements	21
2.11 Use locator name as JSON attribute	22

CONTENTS

2.12	Find multiple elements	22
3.	Hyperlink	23
3.1	Click a link by text	23
3.2	Click a link by ID	23
3.3	Click a link by partial text	24
3.4	Click a link by XPath	24
3.5	Click Nth link with exact same label	25
3.6	Click Nth link by CSS Selector	25
3.7	Verify a link present or not?	25
3.8	Getting link data attributes	26
3.9	Test links open a new browser window	26
4.	Button	28
4.1	Click a button by label	28
4.2	Click a form button by label	28
4.3	Submit a form	29
4.4	Click a button by ID	29
4.5	Click a button by name	30
4.6	Click a image button	30
4.7	Click a button via JavaScript	30
4.8	Assert a button present	31
4.9	Assert a button enabled or disabled?	31
5.	TextField and TextArea	33
5.1	Enter text into a text field by name	33
5.2	Enter text into a text field by ID	33
5.3	Enter text into a password field	34
5.4	Clear a text field	34
5.5	Enter text into a multi-line text area	34
5.6	Assert value	34
5.7	Focus on a control	35
5.8	Set a value to a read-only or disabled text field	35
5.9	Set and assert the value of a hidden field	36
6.	Radio button	37
6.1	Select a radio button	37
6.2	Clear radio option selection	37
6.3	Assert a radio option is selected	38

CONTENTS

6.4	Iterate radio buttons in a radio group	38
6.5	Click Nth radio button in a group	39
6.6	Click radio button by the following label	39
6.7	Customized Radio buttons - iCheck	40
7.	CheckBox	41
7.1	Check by name	41
7.2	Check by id	41
7.3	Uncheck a checkbox	41
7.4	Assert a checkbox is checked (or not)	42
7.5	Customized Checkboxes - iCheck	42
8.	Select List	44
8.1	Select an option by text	44
8.2	Select an option by value	45
8.3	Select an option by iterating all options	45
8.4	Select multiple options	45
8.5	Select options by index	46
8.6	Clear one selection	46
8.7	Clear all selections	47
8.8	Assert a label or value in a select list	47
8.9	Assert selected option label	48
8.10	Assert the value of a select list	48
8.11	Assert multiple selections	49
9.	Navigation and Browser	50
9.1	Go to a URL	50
9.2	Visit pages within a site	50
9.3	Perform actions from right click context menu such as 'Back', 'Forward' or 'Refresh'	51
9.4	Open browser in certain size	51
9.5	Maximize browser window	51
9.6	Move browser window	51
9.7	Minimize browser window	52
9.8	Scroll focus to control	52
9.9	Switch between browser windows or tabs	52
9.10	Remember current web page URL, then come back to it later	53
10.	Assertion	54

CONTENTS

10.1	Assert page title	54
10.2	Assert Page Text	54
10.3	Assert Page Source	55
10.4	Assert Label Text	55
10.5	Assert Span text	55
10.6	Assert Div text or HTML	56
10.7	Assert Table text	57
10.8	Assert text in a table cell	58
10.9	Assert text in a table row	58
10.10	Assert image present	58
10.11	Assert element location and width	58
10.12	Assert element CSS style	59
10.13	Assert JavaScript errors on a web page	59
11.	Frames	61
11.1	Testing Frames	61
11.2	Testing iframe	62
11.3	Test multiple iframes	63
12.	Testing AJAX	64
12.1	Wait within a time frame	65
12.2	Explicit Waits until Time out	65
12.3	Implicit Waits until Time out	66
12.4	Wait AJAX Call to complete using JQuery	66
13.	File Upload and Popup dialogs	68
13.1	File upload	68
13.2	JavaScript pop ups	69
13.3	Timeout on a test	70
13.4	Modal style dialogs	71
13.5	Internet Explorer modal dialog	71
14.	Debugging Test Scripts	73
14.1	Print text for debugging	73
14.2	Write page source or element HTML into a file	73
14.3	Take screenshot	74
14.4	Run single test case with Mocha	75
14.5	Run tests matching a pattern with Mocha	76
14.6	Leave browser open after test finishes	76

CONTENTS

14.7	Run selected test steps against current browser	77
15.	Test Data	80
15.1	Get date dynamically	80
15.2	Get a random boolean value	81
15.3	Generate a number within a range	81
15.4	Get a random character	81
15.5	Get a random string at fixed length	82
15.6	Get a random string in a collection	82
15.7	Generate random person names, emails, addresses with Faker	83
15.8	Generate a test file at fixed sizes	83
15.9	Retrieve data from Database	84
16.	Browser Profile and Capabilities	85
16.1	Get browser type and version	85
16.2	Set HTTP Proxy for Browser	85
16.3	Verify file download in Chrome with Custom User Preferences	86
16.4	Test downloading PDF in Firefox with Custom Profile	87
16.5	Bypass basic authentication by embedding username and password in URL	87
16.6	Bypass basic authentication with Firefox AutoAuth plugin	88
16.7	Manage Cookies	90
16.8	Headless browser testing with PhantomJS	91
16.9	Headless Chrome	92
16.10	Headless Firefox	92
16.11	Test responsive web pages	93
17.	Advanced User Interactions	94
17.1	Double click a control	94
17.2	Move mouse to a control - Mouse Over	95
17.3	Click and hold - select multiple items	95
17.4	Context Click - right click a control	96
17.5	Drag and drop	96
17.6	Drag slider	97
17.7	Send key sequences - Select All and Delete	98
17.8	Click a specific part of an image	99
18.	HTML 5 and Dynamic Web Sites	100
18.1	HTML5 Email type field	100
18.2	HTML5 Time Field	100

CONTENTS

18.3	Invoke ‘onclick’ JavaScript event	101
18.4	Invoke JavaScript events such as ‘onchange’	102
18.5	Scroll to the bottom of a page	102
18.6	Select2 - Single Select	103
18.7	Select2 - Multiple Select	105
18.8	AngularJS web pages	108
18.9	Ember JS web pages	110
18.10	“Share Location” with Firefox	111
18.11	Faking Geolocation with JavaScript	113
18.12	Save a canvas to PNG image	113
18.13	Verify dynamic charts	114
19.	WYSIWYG HTML editors	115
19.1	TinyMCE	115
19.2	CKEditor	116
19.3	SummerNote	118
19.4	CodeMirror	118
20.	Leverage Programming	120
20.1	Raise exceptions to fail test	120
20.2	Ignorable test statement error	122
20.3	Read external file	122
20.4	Data-Driven Tests with CSV	123
20.5	Identify element IDs with dynamically generated long prefixes	124
20.6	Sending special keys such as Enter to an element or browser	124
20.7	Use of Unicode in test scripts	125
20.8	Extract a group of dynamic data : verify search results in order	125
20.9	Verify uniqueness of a set of data	127
20.10	Extract dynamic visible data rows from a results table	127
20.11	Extract dynamic text following a pattern using Regex	129
21.	Optimization	131
21.1	Assert text in page_source is faster than the text	131
21.2	Getting text from more specific element is faster	132
21.3	Avoid programming if-else block code if possible	133
21.4	Enter large text into a text box	133
21.5	Use Environment Variables to change test behaviours dynamically	134
21.6	Testing web site in two languages	135

CONTENTS

22. Gotchas	137
22.1 Test starts browser but no execution with blank screen	137
22.2 Failed to assert copied text in browser	138
22.3 The same test works for Chrome, but not IE	139
22.4 “unexpected tag name ‘input’”	140
22.5 Element is not clickable or not visible	141
23. Material Design Web App	142
23.1 Select List (dropdown)	142
23.2 Checkbox	142
23.3 Drag range (noUiSlider)	143
23.4 Verify Toast message	143
23.5 Modal	144
24. Selenium Remote Control Server	145
24.1 Selenium Server Installation	145
24.2 Execute tests in specified browser on another machine	146
24.3 Selenium Grid	147
25. Quiz	150
25.1 Answers	157
Appendix - Continuous Testing	160
Verify server machine can run Selenium Mocha	160
Install BuildWise Server	161
Create Build Project in BuildWise	162
Trigger test execution manually	163
Feedback while test execution in progress	163
Build finished	165
Notification	166
Review	166
Afterword	167
Resources	169
Books	169
Web Sites	170
Blog	170
Tools	171

Preface

Selenium WebDriver comes with five core language bindings: Java, C#, Ruby, Python and JavaScript (Node.js). I have written Selenium WebDriver recipes for all other four languages except JavaScript. This is not because I don't know JavaScript, as a matter of fact, I have a long history of using JavaScript and I am still using it to develop dynamic web applications. JavaScript, to me, seems always associated with Web and HTML, rather than being a standalone scripting language. One day, overcoming my prejudice, I gave Selenium WebDriver with Node.js a go. I was very impressed. I found Selenium WebDriver in JavaScript is lightweight, quick setup, and above all, test execution is fast, really fast.

Over the years, JavaScript has evolved far beyond a client-side scripting language, and has become a powerful programming language which can also be used to create server-side applications, or even desktop applications (Microsoft's new programmer's editor Visual Studio Code is built with Node.js). Node.js is a cross-platform runtime environment that uses open-source V8 JavaScript Engine execute JavaScript code as an application. As you have probably noticed, the demand for JavaScript programmers is high (see [JavaScript developer salary graph in US](#)¹). It comes to no surprises that Selenium WebDriver includes JavaScript as one of the five official language bindings.

The purpose of this book is to help motivated testers work better with Selenium WebDriver. The book contains over 170 recipes for web application tests with Selenium WebDriver in JavaScript. If you have read my other book: *Practical Web Test Automation*², you probably know my style: practical. I will let the test scripts do most of the talking. These recipe test scripts are 'live', as I have created the target test site and included offline test web pages. With both, you can:

1. **Identify** your issue
2. **Find** the recipe
3. **Run** the test case
4. **See** test execution in your browser

¹<http://www.indeed.com/salary/q-Javascript-Developer-l-United-States.html>

²<https://leanpub.com/practical-web-test-automation>

Who should read this book

This book is for testers or programmers who are writing (or want to learn) automated tests with Selenium WebDriver. In order to get the most of this book, basic (very basic) JavaScript skills is required.

How to read this book

Usually, a ‘recipe’ book is a reference book. Readers can go directly to the part that interests them. For example, if you are testing a multiple select list and don’t know how, you can look up in the Table of Contents, then go to the chapter 8. This book supports this style of reading.

If you are new to Selenium WebDriver, I recommend you to try out the recipes from the front to back. The recipes in the first half of the book are arranged according to their levels of complexity, I believe readers can get the pattern of testing with Selenium and gain confidence after going through them.

Recipe test scripts

To help readers to learn more effectively, this book has a [dedicated site](#)³ that contains the recipe test scripts, test web pages and related resources. For access code, please see the [Resources](#) section of this book.

As an old saying goes, “*There’s more than one way to skin a cat.*” You can achieve the same testing outcome with test scripts implemented in different ways. The recipe test scripts in this book are written for simplicity, and there is always room for improvement. But for many, to understand the solution quickly and get the job done are probably more important.

If you have a better and simpler way, please let me know.

All recipe test scripts are Selenium WebDriver 3 compliant, and can be run against Chrome, Firefox and Internet Explorer on multiple platforms. I plan to keep the test scripts updated with the latest stable Selenium version.

³<http://zhimin.com/books/selenium-recipes-nodejs>

Send me feedback

I would appreciate your comments, suggestions, reports on errors in the book and the recipe test scripts. You may submit your feedback on the book site.

Zhimin Zhan

Brisbane, Australia

1. Introduction

Selenium is a free and open source library for automated testing web applications. Selenium was originally created in 2004 by Jason Huggins, it merged with another test framework WebDriver in 2011 (that's why is named 'selenium-webdriver') led by Simon Stewart at Google (update: Simon now works at FaceBook). As WebDriver is a [W3C standard](https://www.w3.org/TR/webdriver/)¹, it gains support from all major browser vendors, as a result, Selenium WebDriver quickly become the de facto framework for automated testing web applications.

1.1 Selenium language bindings

Selenium tests can be written in multiple programming languages such as Java, C#, Python, Ruby and JavaScript (the core ones). All examples in this book are written in Selenium with JavaScript binding. As you will see from the examples below, the use of Selenium in different bindings are very similar. Once you master one, you can apply it to others quite easily. Take a look at a simple Selenium test script in five different language bindings: JavaScript, Java, C#, Python and Ruby.

JavaScript:

```
var webdriver = require('selenium-webdriver');
var driver = new webdriver.Builder()
    .forBrowser('chrome')
    .build();

driver.get('http://www.google.com/ncr');
driver.findElement(webdriver.By.name('q')).sendKeys('webdriver');
driver.findElement(webdriver.By.name('btnG')).click();
driver.wait(webdriver.until.titleIs('webdriver - Google Search'), 1000);
driver.quit();
```

Java:

¹<https://www.w3.org/TR/webdriver/>

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

public class GoogleSearch {
    public static void main(String[] args) {
        WebDriver driver = new FirefoxDriver();
        driver.get("http://www.google.com");
        WebElement element = driver.findElement(By.name("q"));
        element.sendKeys("Hello Selenium WebDriver!");
        element.submit();
        System.out.println("Page title is: " + driver.getTitle());
    }
}
```

C#:

```
using System;
using OpenQA.Selenium;
using OpenQA.Selenium.Firefox;
using OpenQA.Selenium.Support.UI;

class GoogleSearch
{
    static void Main()
    {
        IWebDriver driver = new FirefoxDriver();
        driver.Navigate().GoToUrl("http://www.google.com");
        IWebElement query = driver.FindElement(By.Name("q"));
        query.SendKeys("Hello Selenium WebDriver!");
        query.Submit();
        Console.WriteLine(driver.Title);
    }
}
```

Python:

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.get("http://www.google.com")

elem = driver.find_element_by_name("q")
elem.send_keys("Hello WebDriver!")
elem.submit()

print(driver.title)
```

Ruby:

```
require "selenium-webdriver"

driver = Selenium::WebDriver.for :firefox
driver.navigate.to "http://www.google.com"

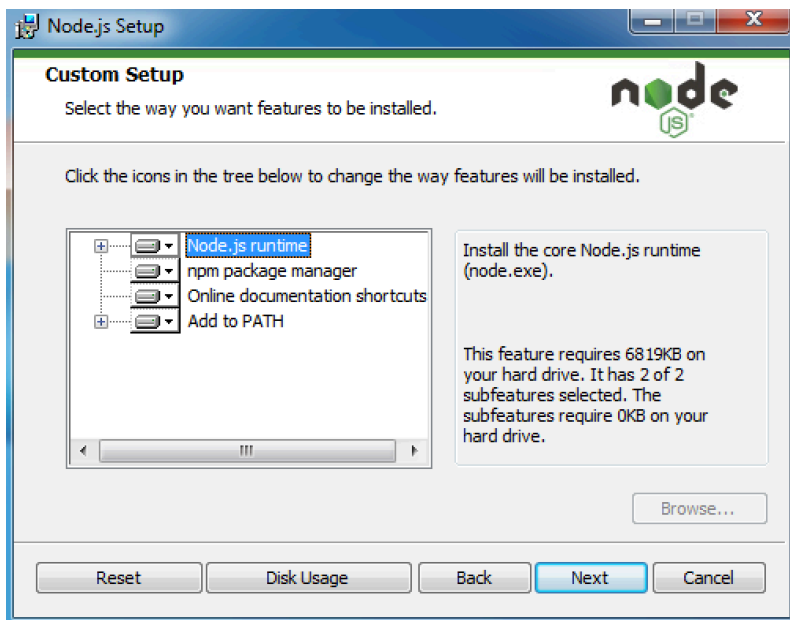
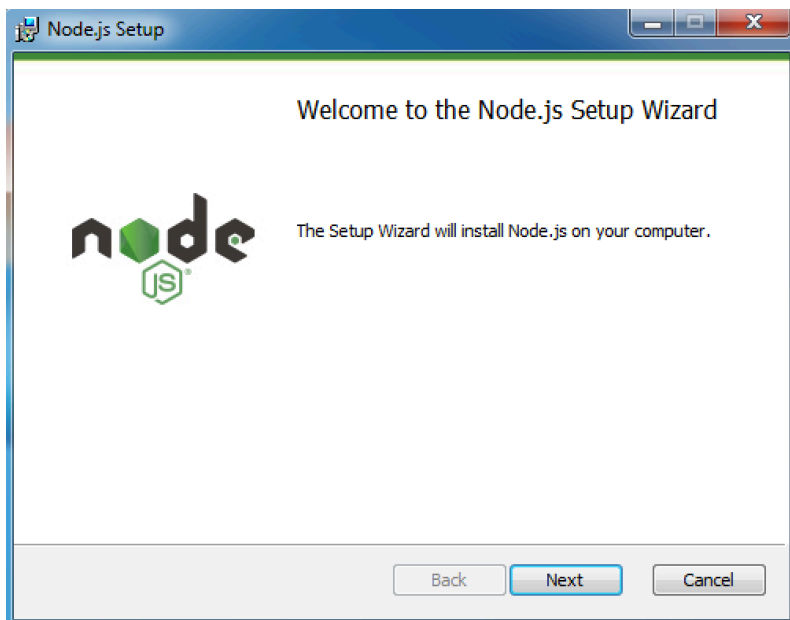
element = driver.find_element(:name, 'q')
element.send_keys "Hello Selenium WebDriver!"
element.submit

puts driver.title
```

1.2 Install Node.JS

[Node.js](https://nodejs.org)² is a JavaScript runtime built on Chrome's V8 JavaScript engine. I use the latest Node.js stable version 10 for the recipes in this book.

²<https://nodejs.org>



Verify **node** and **npm** (Node.js' package manager) are installed:

```
> node --version
v10.7.0
> npm --version
6.4.0
```

1.3 Write first script with JavaScript editor

Many programmer's editors (such as Sublime Text) and IDEs (such as WebStorm) support JavaScript, some even provide specific support for Node.js. In this book, I will use [Visual Studio Code](https://code.visualstudio.com)³, a free source code editor developed by Microsoft (as a matter of fact, Visual Studio Code is developed with Node.js). Visual Studio Code runs on Windows, Linux and OS X.

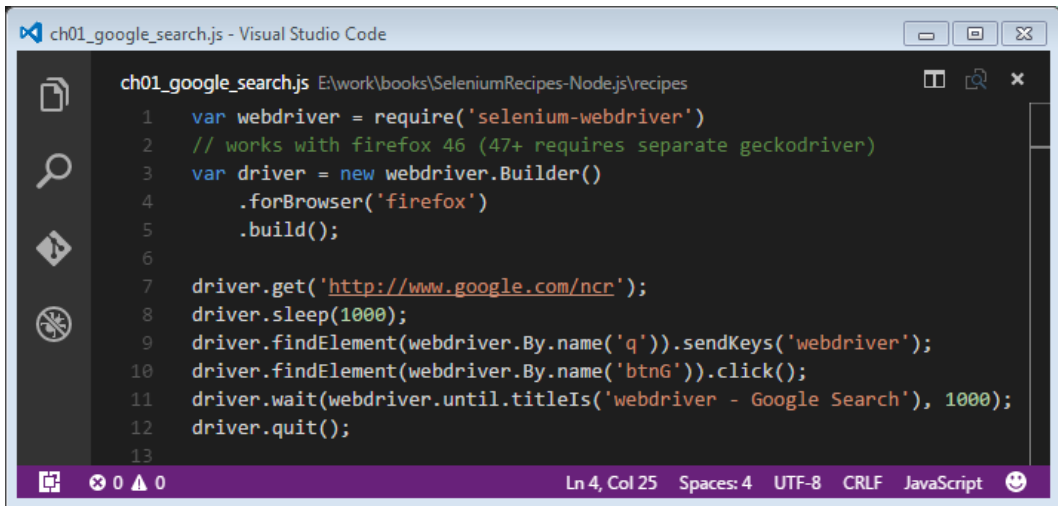
Now let's write one Selenium WebDriver script in Node.js. Get your JavaScript editor or IDE ready, type or paste in the script below:

```
var webdriver = require('selenium-webdriver')
var driver = new webdriver.Builder()
    .forBrowser('chrome')
    .build();

driver.get('http://www.google.com/ncr');
driver.sleep(1000);
driver.findElement(webdriver.By.name('q')).sendKeys('webdriver');
driver.findElement(webdriver.By.name('btnG')).click();
driver.wait(webdriver.until.titleIs('webdriver - Google Search'), 1000);
driver.quit();
```

Save the file as *ch01_google_search.js*. It will look like this in Visual Studio Code:

³<https://code.visualstudio.com>



```
ch01_google_search.js E:\work\books\SeleniumRecipes-Node.js\recipes
1  var webdriver = require('selenium-webdriver')
2  // works with firefox 46 (47+ requires separate geckodriver)
3  var driver = new webdriver.Builder()
4    .forBrowser('firefox')
5    .build();
6
7  driver.get('http://www.google.com/ncr');
8  driver.sleep(1000);
9  driver.findElement(webdriver.By.name('q')).sendKeys('webdriver');
10 driver.findElement(webdriver.By.name('btnG')).click();
11 driver.wait(webdriver.until.titleIs('webdriver - Google Search'), 1000);
12 driver.quit();
13
```

1.4 Install Selenium WebDriver

By default, Node.js scripts load dependent (npm) packages installed locally, i.e, the packages are installed under the directory where the scripts located.

```
> cd YOUR_SCRIPT_DIRECTORY
> npm install selenium-webdriver@3.6.0
```

When it is done, you will see output like below:

```
+ selenium-webdriver@3.6.0
```

At the time of writing, the latest release of Selenium WebDriver for JavaScript is 4.0.0-alpha.7, which is a pre-release version (and I found some issues with it). For some unknown reason, `npm install selenium-webdriver` will install 4.0.0-alpha.7 as a stable version. So here I use the syntax for installing a specific version of a npm module.

This will create the **node_modules** directory in your current directory, and the packages will be installed into that directory.

If you run into issues on npm package installation, refer [NPM doc](https://docs.npmjs.com/getting-started/installing-npm-packages-locally)⁴.

⁴<https://docs.npmjs.com/getting-started/installing-npm-packages-locally>

1.5 Run script

Prerequisite:

- Chrome browser with ChromeDriver installed (see below)

Start a new command (or terminal) window, change to the the script directory, run the command:

```
> node ch01_google_search.js
```

You shall see a new Chrome window open, perform a Google search of ‘webdriver’ and close the browser.

1.6 Cross browser testing

The biggest advantage of Selenium over other web test frameworks, in my opinion, is that it supports all major web browsers: Firefox, Chrome and Internet Explorer/Edge. The browser market nowadays is more diversified (based on the [StatsCounter](https://www.statscounter.com/)⁵, the usage share in June 2020 for Chrome, Firefox, Safari, IE/Edge and are 69.4%, 8.5%, 8.7% and 8.7% respectively). It is logical that all external facing web sites require serious cross-browser testing. Selenium is a natural choice for this purpose, as it far exceeds other commercial tools and free test frameworks.


Chrome

To run Selenium tests in Google Chrome, besides the Chrome browser itself, *ChromeDriver* needs to be installed.





Installing ChromeDriver is easy: go to <http://chromedriver.storage.googleapis.com/index.html>⁶

⁵http://en.wikipedia.org/wiki/Usage_share_of_web_browsers

⁶<http://chromedriver.storage.googleapis.com/index.html>

← → ↻  chromedriver.storage.googleapis.com/index.html?path=80.0.3987.106/

Index of /80.0.3987.106/

Name	Last modified	Size	ETag
 Parent Directory		-	
 chromedriver_linux64.zip	2020-02-13 19:21:31	4.71MB	caf2eb7148c03617f264b99743e2051c
 chromedriver_mac64.zip	2020-02-13 19:21:32	6.68MB	675a673c111fdcc9678d11df0e69b334
 chromedriver_win32.zip	2020-02-13 19:21:34	4.17MB	d5fee78fdbcb9c2c3af9a2ce1299a8621

Download the one for your target platform, unzip it and put **chromedriver** executable in your PATH. To verify the installation, open a command window (terminal for Unix/Mac), execute command *chromedriver*, You shall see:

```
Starting ChromeDriver 80.0.3987.16 (...) on port 9515
Only local connections are allowed.
Please protect ports used by ChromeDriver and related test frameworks to prevent
access by malicious code.
```

The test script below opens a site in a new Chrome browser window and closes it one second later.

```
var webdriver = require('selenium-webdriver');
var driver = new webdriver.Builder()
    .forBrowser('chrome')
    .build();
```

Firefox

Selenium tests requires [Gecko Driver](https://github.com/mozilla/geckodriver/releases/)⁷ to drive Firefox. The test script below will open a web site in a new Firefox window.

```
var webdriver = require('selenium-webdriver');
var driver = new webdriver.Builder()
    .forBrowser('firefox')
    .build();
```

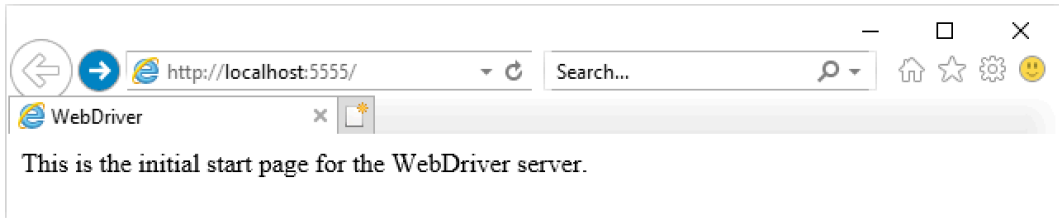
⁷<https://github.com/mozilla/geckodriver/releases/>

Internet Explorer

Selenium requires IEDriverServer to drive IE browser. Its installation process is very similar to *ChromeDriver*. IEDriverServer is available at <http://www.seleniumhq.org/download/>⁸. Choose the right one based on your windows version (32 or 64 bit).

Download version 3.14.0 for (recommended) [32 bit Windows IE](#) or [64 bit Windows IE](#)
[CHANGELOG](#)

When a tests starts to execute in IE, before navigating the target test site, you will see this:



Depending on the version of IE, configurations may be required. Please see [IE and IEDriverServer Runtime Configuration](#)⁹ for details.

```
var webdriver = require('selenium-webdriver');  
var driver = new webdriver.Builder()  
    .forBrowser('ie')  
    .build();
```

Edge Chromium

Edge (Chromium) is Microsoft's new and default web browser that works on all common platforms, including macOS. An additional library 'edge-selenium-tools' (besides 'selenium-webdriver') is required.

```
npm install @microsoft/edge-selenium-tools
```

The driver for Edge Chromium is 'msedgedriver', which you can download at <https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/>¹⁰.

⁸<http://www.seleniumhq.org/download/>

⁹https://code.google.com/p/selenium/wiki/InternetExplorerDriver#Required_Configuration

¹⁰<https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/>

```
var webdriver = require('selenium-webdriver');
var edge = require('selenium-webdriver/edge');

let options = new edge.Options();
options.setEdgeChromium(true);
let driver = edge.Driver.createSession(options);
driver.get("https://whenwise.agileway.net");
```

Edge Chromium is very similar to Chrome, as they are both based on the Chromium. Unless there is a specific requirement for Edge, I would suggest using Google Chrome for running selenium tests for stability (*ChromeDriver has been around since the initial release of Selenium WebDriver*).

Edge Legacy

To drive Edge (Legacy) with WebDriver, you need download [Microsoft WebDriver](#)¹¹. After installation, you will find the executable (*MicrosoftWebDriver.exe*) under *Program Files* folder, add it to your PATH.

However, I couldn't get it working after installing a new version of Microsoft WebDriver. One workaround is to specify the driver path in test scripts specifically:

```
// copy MicrosoftWebDriver.exe to the test script directory
var webdriver = require('selenium-webdriver');
var edge = require('selenium-webdriver/edge');
var service = new edge.ServiceBuilder(__dirname + '/MicrosoftWebDriver.exe')
    .build();
var options = new edge.Options();
var driver = new edge.Driver(options, service);
driver.get("http://www.google.com/ncr");
```

1.7 Mocha Test Framework

The above scripts drive browsers, strictly speaking, they are not tests. To make the effective use of Selenium scripts for testing, we need to put them in a test syntax framework that defines test structures and provides assertions (performing checks in test scripts). There are several popular JavaScript test frameworks, such as [Mocha](#)¹², [Karma](#)¹³, and [Jasmine](#)¹⁴.

¹¹<https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/>

¹²<https://mochajs.org/>

¹³<https://github.com/karma-runner/karma>

¹⁴<http://jasmine.github.io/>

I use Mocha in this book. You are free to choose any JavaScript test framework, Selenium WebDriver techniques are applicable regardless of syntax frameworks. Here is a Mocha test script for user login:

```
var webdriver = require('selenium-webdriver');
var test = require('selenium-webdriver/testing'); // add 'test.' wrapper
var driver = new webdriver.Builder()
    .forBrowser('chrome')
    .build();
var assert = require('assert');

test.describe('User Authentication', function () {

    test.it('User can sign in', function () {
        driver.get('http://travel.agileway.net');
        driver.findElement(webdriver.By.name('username')).sendKeys('agileway');
        driver.findElement(webdriver.By.name('password')).sendKeys('testwise');
        driver.findElement(webdriver.By.name('commit')).click();
        driver.getTitle().then(function(the_title){
            assert.equal("Agile Travel", the_title);
        });
    });

});
```

The keywords `describe` and `it` define the structure of a Mocha test script. I used `test.describe` and `test.it` here, provided with `selenium-webdriver/testing` module, which transparently waits for the promise to resolve before resuming the function.

- **test.describe**
Description of a collection of related test cases.
- **test.it**
Individual test case.

I use Node.js' [assert module](https://nodejs.org/api/assert.html)¹⁵ here to perform checks: `assert.equal(...)`.

Mocha hooks

If you worked with xUnit before, you must have known `setUp()` and `tearDown()` fixtures that are run before or after every test. Mocha provides similar hooks `before()`, `after()`,

¹⁵<https://nodejs.org/api/assert.html>

`beforeEach()`, and `afterEach()`, which can be used to set up preconditions and clean up after test scripts.

```
test.describe('User Authentication', function () {

  test.before(function() {
    // run before all test cases, commonly initialize WebDriver
    driver = new webdriver.Builder()
      .forBrowser('chrome')
      .build();
  });

  test.beforeEach(function() {
    // run before each test case, typically visit home page
    driver.get("http://travel.agilway.net")
  });

  test.afterEach(function() {
    // run after each test case
  });

  test.after(function() {
    // run after all test cases, typically close browser
    driver.quit();
  });

  test.it('Test case description', function() {
    // one test case

  });

  test.it('Another Test case description', function() {
    // another test case

  });

});
```

Install Mocha

Use `npm` to install Mocha package.

```
> npm install -g mocha
```

`-g` tells npm to install this module globally, i.e, any Node.js project can use this module. When it is done, the output will be like below:

```
+ mocha@8.0.1
```

Run the command below in a new Command window (or Terminal on Mac/Linux) to verify that Mocha is installed.

```
> mocha --version  
8.0.1
```

1.8 Create package.json file

A **package.json** file contains metadata about your Node.js app (in our case, test project), usually in the project root. It includes the list of module dependencies to install from npm. While it is not mandatory, it is a good idea to have it so that it will be easy to install or update the project's dependent modules (just run `npm install`). If you are familiar with Ruby, it is similar to a Gemfile.

To create *package.json*, run the command below in the test project's root folder.

```
> npm init
```

Follow the prompt and answer the questions, the answers will be saved in *package.json* file in the folder. Then install two modules: *selenium-webdriver* and *mocha*. You might have noticed the difference from the previous `npm install` commands: using `--save` instead of `-g`. This tell npm to install modules locally and update the *package.json* file.

```
> npm install --save selenium-webdriver@3.6.0  
> npm install --save mocha
```

After installation, you will see folder **node_modules** (locally installed modules) and the following content in the *package.json* file.

```
"dependencies": {  
  "mocha": "^8.0.1",  
  "selenium-webdriver": "^3.6.0",  
}
```

1.9 Run recipe test scripts

Test scripts for all recipes can be downloaded from this book's site. They are all in a ready-to-run state. I include the target web pages/sites as well as Selenium test scripts. There are two kinds of target web pages: local HTML files and web pages on a live site. An Internet connection is required to run tests written for a live site.

One key advantage of open-source test frameworks, such as Selenium WebDriver and Mocha, is FREEDOM. You can edit the test scripts in any text editors and run them from a command line. For example, To run test cases in a test script file (named *ch01_agiletravel_login_spec.js*), run command

```
> mocha ch01_agiletravel_login_spec.js
```

You might get the error output like below:

Timeout error

User Authentication

1) User can sign in

0 passing (2s)

1 failing

1) User Authentication User can sign in:

Error: timeout of 2000ms exceeded. Ensure the done() callback is being called in this test.

at Timeout.<anonymous> (C:\...\npm\node_modules\mocha\lib\runnable.js:226:19)

This is because the default timeout for a test case (or hook) is 2000 ms (i.e. 2 seconds). One common way is to override the timeout value at the beginning of a test case:

```
test.it('User can sign in', function() {  
  this.timeout(8000);  
  driver.get('http://travel.agileway.net');  
  // ...  
});
```

Run the test script again, it shall pass. Here is a sample output with two test cases in one test script file.

Sample Success Output

User Authentication

- ✓ Invalid user (784ms)
- ✓ User can login successfully (1197ms)

2 passing (11s)

2. Locating web elements

As you might have already figured out, to drive an element in a page, we need to find it first. Selenium WebDriver uses what is called locators to find and match the elements on web page. There are 8 locators in Selenium WebDriver:

Locator	Example
ID	<code>findElement(By.id("user"))</code>
Name	<code>findElement(By.name("username"))</code>
Link Text	<code>findElement(By.linkText("Login"))</code>
Partial Link Text	<code>findElement(By.partialLinkText("Next"))</code>
XPath	<code>findElement(By.xpath("//div[@id='login']/input"))</code>
Tag Name	<code>findElement(By.tagName("body"))</code>
Class Name	<code>findElement(By.className("table"))</code>
CSS	<code>findElement(By.css, "#login > input[type='text']")</code>

You may use any one of them to narrow down the element you are looking for.

2.1 Start browser

Testing web sites starts with a browser. The test script below launches a Chrome browser window and navigate to a site.

```
var webdriver = require('selenium-webdriver');
var driver = new webdriver.Builder()
    .forBrowser('chrome')
    .build();
driver.get("http://testwisely.com/demo")
```

Use `firefox` and `ie` for testing in Firefox and IE respectively.

Test Pages

I prepared the test pages for the recipes, you can download it at [the book's site](#)^a. Unzip it to a local directory and refer to test pages like this:

```
driver.get("file://" + __dirname + "/../../site/locators.html");
```

__dirname is the directory where the test script is.

<http://zhimin.com/books/selenium-recipes-nodejs>

I recommend, for beginners, to close the browser window at the end of a test case.

```
driver.quit();
```

2.2 Find element by ID

Using IDs is the easiest and probably the safest way to locate an element in HTML. If a web page is [W3C HTML conformed](http://www.w3.org/TR/WCAG20-TECHS/H93.html)¹, the IDs should be unique and identified in web controls on the page. In comparison to texts, test scripts that use IDs are less prone to application changes (e.g. developers may decide to change the label, but are less likely to change the ID).

```
var webdriver = require('selenium-webdriver');
var driver = new webdriver.Builder()
    .forBrowser('chrome')
    .build();

// ...
driver.findElement(webdriver.By.id("submit_btn")).click();
```

As we will use locator statement `webdriver.By` frequently in test scripts, we usually create a shorthand `By` for it as below.

¹<http://www.w3.org/TR/WCAG20-TECHS/H93.html>

```
var webdriver = require('selenium-webdriver'),
    By = webdriver.By,
    until = webdriver.until;
var driver = new webdriver.Builder()
    .forBrowser('chrome')
    .build();

//...
driver.findElement(By.id("cancel_link")).click();           // Link
driver.findElement(By.id("username")).sendKeys("agileway"); // Textfield
driver.findElement(By.id("alert_div")).getText();           // HTML Div element
```

2.3 Find element by Name

The name attributes are used in form controls such as text fields and radio buttons. The values of the name attributes are passed to the server when a form is submitted. In terms of least likelihood of a change, the name attribute is probably only second to ID.

```
driver.findElement(By.name("comment")).sendKeys("Selenium Cool");
```

2.4 Find element by Link Text

For Hyperlinks only. Using a link's text is probably the most direct way to click a link, as it is what we see on the page.

```
driver.findElement(By.linkText("Cancel")).click();
```

2.5 Find element by Partial Link Text

Selenium allows you to identify a hyperlink control with a partial text. This can be quite useful when the text is dynamically generated. In other words, the text on one web page might be different on your next visit. We might be able to use the common text shared by these dynamically generated link texts to identify them.

```
// will click the "Cancel" link
driver.findElement(By.partialLinkText("ance")).click();
```

2.6 Find element by XPath

XPath, the XML Path Language, is a query language for selecting nodes from an XML document. When a browser renders a web page, it parses it into a DOM tree or similar. XPath can be used to refer a certain node in the DOM tree. If this sounds too technical for you, don't worry, just remember XPath is the most powerful way to find a specific web control.

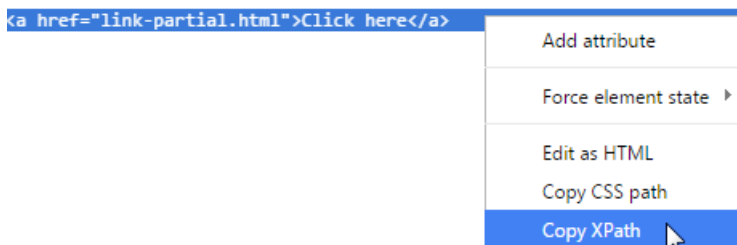
```
// clicking the checkbox under 'div2' container
driver.findElement(By.xpath("//*[@id='div2']/input[@type='checkbox']")).click();
```

Some testers feel intimidated by the complexity of XPath. However, in practice, there is only limited scope of XPath to master for testers.



Avoid using copied XPath from Browser's Developer Tool

Browser's Developer Tool (right click to select 'Inspect element' to show) is very useful for identifying a web element in web page. You may get the XPath of a web element there, as shown below (in Chrome):



The copied XPath for the second "Click here" link in the example:

```
//*[@id="container"]/div[3]/div[2]/a
```

It works. However, I do not recommend this approach as the test script is fragile. If developer adds another `div` under `<div id='container'>`, the copied XPath is no longer correct for the element while `//div[contains(text(), "Second")]/a[text()='Click here']` still works.

In summary, XPath is a very powerful way to locating web elements when `id`, `name` or `linkText` are not applicable. Try to use a XPath expression that is less vulnerable to structure changes around the web element.

2.7 Find element by Tag Name

There are a limited set of tag names in HTML. In other words, many elements share the same tag names on a web page. We normally don't use the `tagName` locator by itself to locate an element. We often use it with others in a chained locators (see the section below). However, there is an exception.

```
driver.findElement(By.tagName("body")).getText();
```

The above test statement returns the text view of a web page, this is a very useful one as Selenium WebDriver does not have built-in method return the text of a web page. However, unique this JavaScript binding, we cannot use this way directly. For example, the script below prints out the body's text.

```
var pageText = driver.findElement(By.tagName("body")).getText();
console.log(pageText);
```

The output is a `ManagedPromise`.

```
ManagedPromise {
  flow_:
    ControlFlow {
      propagateUnhandledRejections_: true,
      ...
    }
}
```

What is a Promise?

A Promise is “an object that represents a value, or the eventual computation of a value”. Essentially, a promise is a result of async operation that is not resolved yet. If you are not familiar with JavaScript, it can be quite confusing. Good news is that, for the context of Selenium WebDriver, you will soon find out the pattern when there is need to resolve a promise.

To resolve a promise, use `.then(function(returnedValue)){ ... }.`

```
driver.findElement(By.tagName("body")).getText().then(function(body_text){
    console.log(body_text)
});
```

2.8 Find element by Class

The `class` attribute of a HTML element is used for styling. It can also be used for identifying elements. Commonly, a HTML element's class attribute has multiple values, like below.

```
<a href="back.html" class="btn btn-default">Cancel</a>
<input type="submit" class="btn btn-default btn-primary">Submit</input>
```

You may use any one of them.

```
driver.findElement(By.className("btn-primary")).click(); // Submit button
driver.findElement(By.className("btn")).click();          // Cancel link
```

The `className` locator is convenient for testing JavaScript/CSS libraries (such as TinyMCE) which typically use a set of defined class names.

```
// inline editing
driver.findElement(By.id("client_notes")).click();
driver.sleep(500);
driver.findElement(By.className("editable-textarea")).sendKeys("inline");
driver.sleep(500);
driver.findElement(By.className("editable-submit")).click();
```

2.9 Find element by CSS

You may also use CSS Path to locate a web element.

```
driver.findElement(By.css("#div2 > input[type='checkbox']")).click();
```

However, the use of CSS is generally more prone to structure changes of a web page.

2.10 Chain `findElement` to find child elements

For a page containing more than one elements with the same attributes, like the one below, we could use XPath locator.

```

<div id="div1">
  <input type="checkbox" name="same" value="on"> Same checkbox in Div 1
</div>
<div id="div2">
  <input type="checkbox" name="same" value="on"> Same checkbox in Div 2
</div>

```

There is another way: chain `findElement` to find a child element.

```
driver.findElement(By.id("div2")).findElement(By.name("same")).click();
```

2.11 Use locator name as JSON attribute

You may use locator as JSON attribute as below.

```

driver.findElement({name: 'comment'}).sendKeys("JSON")    // a text box
driver.findElement({css: "#div1 > input[type='checkbox']"}).click();
driver.findElement({className: "btn-primary"}).click();    // Submit button

```

2.12 Find multiple elements

As its name suggests, `findElements` return a list of matched elements back. Its syntax is exactly the same as `findElement`, i.e. can use any of 8 locators.

The test statements will find two checkboxes under `div#container` and click the second one.

```

let xpathStr = "//div[@id='container']/input[@type='checkbox']";
var checkboxElems = driver.findElements(By.xpath(xpathStr))
driver.sleep(500)
// console.log("XXX: " + checkboxElems.then.length + "\n"); // 2
driver.findElements(By.xpath(xpathStr)).then(function(checkboxElems) {
  checkboxElems[1].click(); // second one
});

```

Sometimes `findElement` fails due to multiple matching elements on a page, which you were not aware of. `findElements` will come in handy to find them out.

3. Hyperlink

Hyperlinks (or links) are fundamental elements of web pages. As a matter of fact, it is hyperlinks that makes the World Wide Web possible. A sample link is provided below, along with the HTML source.

[Recommend Selenium](#)

HTML Source

```
<a href="index.html" id="recommend_selenium_link" class="nav" data-id="123" style="font-size: 14px;">Recommend Selenium</a>
```

3.1 Click a link by text

Using text is probably the most direct way to click a link in Selenium, as it is what we see on the page.

```
// driver.get("file://" + __dirname + "/../../site/link.html");  
driver.findElement(By.linkText("Recommend Selenium")).click();
```

3.2 Click a link by ID

```
driver.findElement(By.id("recommend_selenium_link")).click();
```

Furthermore, if you are testing a web site with multiple languages, using IDs is probably the only feasible option. You do not want to write test scripts like below:

```
if (isItalian()) {
    driver.findElement(By.linkText("Accedi")).click();
} else if (isChinese()) { // a helper function determines the locale
    driver.findElement(By.linkText, "☞").click();
} else {
    driver.findElement(By.linkText("Sign in")).click();
}
```

3.3 Click a link by partial text

```
driver.findElement(By.partialLinkText("Recommend Seleni")).click();
```

3.4 Click a link by XPath

The example below is finding a link with text ‘Recommend Selenium’ under a <p> tag.

```
driver.findElement(By.xpath( "//p/a[text()='Recommend Selenium']")).click();
```

Your might say the example before (find by linkText) is simpler and more intuitive, that’s correct. but let’s examine another example:

First div [Click here](#)

Second div [Click here](#)

On this page, there are two ‘Click here’ links.

HTML Source

```
<div>
  First div
  <a href="link-url.html">Click here</a>
</div>
<div>
  Second div
  <a href="link-partial.html">Click here</a>
</div>
```

If test case requires you to click the second ‘Click here’ link, findElement(By.linkText("Click here")) won’t work (as it clicks the first one). Here is a way to accomplish using XPath:

```
driver.findElement(By.xpath('//div[contains(text(), "Second")]/a[text()="Click here"]')).\
click();
```

3.5 Click Nth link with exact same label

It is not uncommon that there are more than one link with exactly the same text. By default, Selenium will choose the first one. What if you want to click the second or Nth one?

The web page below contains two ‘Click here’ links,

[Same link](#) [Different link](#) [Same link](#)

To click the second one,

```
driver.findElements(By.linkText("Click here")).then(function(the_same_links){
    assert.equal(2, the_same_links.length);
    the_same_links[1].click(); // second link
});
```

`findElements` return a list (also called array) of web controls matching the criteria in appearing order. Selenium (in fact JavaScript) uses 0-based indexing, i.e., the first one is 0.

3.6 Click Nth link by CSS Selector

You may also use CSS selector to locate a web element.

```
driver.findElement(By.css("p > a:nth-child(3)")).click(); // 3rd link
```

However, generally speaking, the stylesheet are more prone to changes.

3.7 Verify a link present or not?

```
assert(driver.findElement(By.linkText("Recommend Selenium")).isDisplayed())
assert(driver.findElement(By.id("recommend_selenium_link")).isDisplayed())
```

3.8 Getting link data attributes

Once a web control is identified, we can get its other attributes of the element. This is generally applicable to most of the controls.

```
driver.findElement(By.linkText("Recommend Selenium")).getAttribute("href").then(
    function(the_href) {
        assert(the_href.contains("/site/index.html"))
    });
driver.findElement(By.linkText("Recommend Selenium")).getAttribute("id").then(function(th\
e_id) {
    assert.equal("recommend_selenium_link", the_id)
});
driver.findElement(By.id("recommend_selenium_link")).getText().then(
    function(elemtext){
        assert.equal("Recommend Selenium", elemtext)
    });
driver.findElement(By.id("recommend_selenium_link")).getTagName().then(
    function(tagname) {
        assert.equal("a", tagname)
    });
```

Also you can get the value of custom attributes of this element and its inline CSS style.

```
driver.findElement(By.id("recommend_selenium_link")).getAttribute("style").then(
    function(the_style){
        assert.equal("font-size: 14px;", the_style)
    });

driver.findElement(By.id("recommend_selenium_link")).getAttribute("data-id").then(
    function(the_data_id){
        assert.equal("123", the_data_id)
    });
```

3.9 Test links open a new browser window

Clicking the link below will open the linked URL in a new browser window or tab.

```
<a href="http://testwisely.com/demo" target="_blank">Open new window</a>
```

While we could use `switchTo()` method (see chapter 9) to find the new browser window, it will be easier to perform all testing within one browser window. Here is how:

```
var currentUrl = driver.getCurrentUrl();
new_window_url = driver.findElement(By.linkText("Open new window")).getAttribute("href")
driver.get(new_window_url)
// ... testing on new site
driver.findElement(By.name("name")).sendKeys("sometext")
driver.get(currentUrl) // back
```

In this test script, we use a local variable 'currentUrl' to store the current URL.

4. Button

Buttons can come in two forms - standard and submit buttons. Standard buttons are usually created by the 'button' tag, whereas submit buttons are created by the 'input' tag (normally within form controls).

Standard button

Choose Selenium

Submit button in a form

Username:

HTML Source

```
<button id="choose_selenium_btn" class="nav" data-id="123" style="font-size: 14px;">
  Choose Selenium</button>
<!-- ... -->
<form name="input" action="index.html" method="get">
  Username: <input type="text" name="user">
  <input type="submit" name="submit_action" value="Submit">
</form>
```

Please note that some controls look like buttons, but are actually hyperlinks by CSS styling.

4.1 Click a button by label

```
// driver.get("file://" + __dirname + "../..../site/button.html");

driver.findElement(By.xpath("//button[contains(text(),'Choose Selenium')]")).click();
```

4.2 Click a form button by label

For an input button (in a HTML input tag) in a form, the text shown on the button is the 'value' attribute which might contain extra spaces or invisible characters.

```
<input type="submit" name="submit_action" value="Space After " />
```

The test script below will fail as there is a space character in the end.

```
driver.findElement(By.xpath("//input[@value='Space After']")).click();
```

Changing to match the value exactly will fix it.

```
driver.findElement(By.xpath("//input[@value='Space After ']")).click();
```

4.3 Submit a form

In the official Selenium tutorial, the operation of clicking a form submit button is done by calling *submit* function on an input element within a form. For example, the test script below is to test user sign in.

```
driver.findElement(By.name("user")).sendKeys("agileway");  
password_element = driver.findElement(By.name("password"));  
password_element.sendKeys("secret");  
password_element.submit();
```

However, this is not my preferred approach. Whenever possible, I write test scripts this way: one test step corresponds to one user operation, such as a text entry or a mouse click. This helps me to identify issues quicker during test debugging. Using *submit* means testers need a step to define a variable to store an identified element (line 1 in above test script), to me, it breaks the flow. Here is my version:

```
driver.findElement(By.name("user")).sendKeys("agileway");  
driver.findElement(By.name("password")).sendKeys("secret");  
driver.findElement(By.xpath("//input[@value='Sign in']")).click();
```

Furthermore, if there is more than one submit button (unlikely but possible) in a form, calling *submit* is equivalent to clicking the first submit button only, which might cause confusion.

4.4 Click a button by ID

As always, a better way to identify a button is to use IDs. This applies to all controls, if there are IDs present.

```
driver.findElement(By.id("choose_selenium_btn")).click();
```



For testers who work with the development team, rather than spending hours finding a way to identify a web control, just go to programmers and ask them to add IDs. It usually takes very little effort for programmers to do so.

4.5 Click a button by name

In an input button, we can use a new generic attribute name to locate a control.

```
driver.findElement(By.name("submit_action")).click();
```

4.6 Click a image button

There is also another type of 'button': an image that works like a submit button in a form.



```
<input type="image" src="images/button_go.jpg"/>
```

Besides using ID, the button can also be identified by using `src` attribute.

```
driver.findElement(By.xpath("//input[contains(@src, 'button_go.jpg')]")).click();
```

4.7 Click a button via JavaScript

You may also invoke clicking a button via JavaScript. I had a case where normal approaches didn't click a button reliably on Firefox, but this Javascript way worked well.

```
the_btn = driver.findElement(By.id("searchBtn"));  
driver.executeScript("arguments[0].click();", the_btn);
```

The state of a button can be controlled by JavaScript as well, such as:

```
driver.executeScript("arguments[0].disabled = true;", the_btn);
```

4.8 Assert a button present

Just like hyperlinks, we can use `isDisplayed()` to check whether a control is present on a web page.

```
assert(driver.findElement(By.id("choose_selenium_btn")).isDisplayed());
driver.findElement(By.linkText("Hide")).click().then(function(){
    driver.findElement(By.id("choose_selenium_btn")).isDisplayed().then(function(displayed){
        assert(!displayed);
    });
});
driver.findElement(By.linkText("Show")).click().then(function(){
    driver.findElement(By.id("choose_selenium_btn")).isDisplayed().then(function(displayed){
        assert(displayed);
    });
})
```

This check applies to most of the web controls, such as HyperLinks, Text Fields, ..., etc.

```
driver.findElement(By.linkText("Show")).isDisplayed().then(function(displayed) {
    assert(displayed);
});
```

4.9 Assert a button enabled or disabled?

A web control can be in a disabled state. A disabled button is un-clickable, and it is displayed differently.



Choose Selenium

```
assert(driver.findElement(By.id("choose_selenium_btn")).isEnabled());
driver.findElement(By.linkText("Disable")).click().then(function(){
    driver.findElement(By.id("choose_selenium_btn")).isEnabled().then(function(enabled){
        assert(!enabled);
    });
})
driver.findElement(By.linkText("Enable")).click().then(function(){
    driver.findElement(By.id("choose_selenium_btn")).isEnabled().then(function(enabled){
        assert(enabled);
    });
});
```

i> Normally, enabling or disabling buttons (or other web controls) is triggered by JavaScripts, such as `$('#choose_selenium_btn').removeAttr('disabled')` in JQuery.

5. TextField and TextArea

Text fields are commonly used in a form to pass user entered text data to the server. There are two variants (prior to HTML5): password fields and text areas. The characters in password fields are masked (shown as asterisks or circles). Text areas allows multiple lines of texts.

Username:

Password:

Comments:

HTML Source

```
Username: <input type="text" name="username" id="user"><br>
Password: <input type="password" name="password" id="pass"> <br/>
Comments: <br/>
<textarea id="comments" rows="2" cols="60" name="comments"></textarea>
```

5.1 Enter text into a text field by name

Selenium's `sendKeys` 'types' characters (*including invisible ones such Enter key, see a recipe in Chapter 20*) into a text box.

```
// driver.get("file://" + __dirname + "/../../site/text_field.html");
driver.findElement(By.name("username")).sendKeys("agileway");
```

The 'name' attribute is the identification used by the programmers to process data. It applies to all the web controls in a standard web form.

5.2 Enter text into a text field by ID

```
driver.findElement(By.id("user")).sendKeys("agileway");
```

5.3 Enter text into a password field

In Selenium, password text fields are treated as normal text fields, except that the entered text is masked.

```
driver.findElement(By.id("pass")).sendKeys("testisfun");
```

5.4 Clear a text field

Calling `sendKeys()` to the same text field will concatenate the new text with the old text. So it is a good idea to clear a text field first, then send keys to it.

```
driver.findElement(By.name("username")).sendKeys("test");  
driver.findElement(By.name("username")).sendKeys(" wisely");  
// now => 'test wisely'  
driver.findElement(By.name("username")).clear();  
driver.findElement(By.name("username")).sendKeys("agileway");
```

5.5 Enter text into a multi-line text area

Selenium treats text areas the same as text fields.

```
driver.findElement(By.id("comments")).sendKeys("Automated testing is\r\nFun!");
```

The `\r\n` represents a new line.

5.6 Assert value

```
driver.findElement(By.id( "user")).sendKeys("testwisely");
driver.findElement(By.id("user")).getAttribute("value").then(function(value){
    assert.equal("testwisely", value);
});
```

5.7 Focus on a control

Once we identify one control, we can set the focus on it. There is no *focus* function on *element* in Selenium, we can achieve ‘focusing a control’ by sending empty keystrokes to it.

```
driver.findElement(By.id("pass")).sendKeys("");
```

Or using JavaScript.

```
the_elem = driver.findElement(By.id( "pass"));
driver.executeScript("arguments[0].focus();", the_elem);
```

This workaround can be quite useful. When testing a long web page and some controls are not visible, trying to click them might throw “Element is not visible” error. In that case, setting the focus on the element might make it a visible.

5.8 Set a value to a read-only or disabled text field

‘Read only’ and ‘disabled’ text fields are not editable and are shown differently in the browser (typically grayed out).

Read only text field:

```
<input type="text" name="readonly_text" readonly="true"/> <br/>
```

Disabled text field:

```
<input type="text" name="disabled_text" disabled="true"/>
```

If a text box is set to be read-only, the following test step will not work.

```
driver.findElement(By.name("readonly_text")).sendKeys("new value");
```

Here is a workaround:


```
driver.executeScript("$('#readonly_text').val('bypass');")
driver.findElement(By.id( "readonly_text")).getAttribute("value").then(
    function(the_value){
        assert.equal("bypass", the_value);
    });
```

The below is a screenshot of a disabled and read-only text fields that were 'injected' with two values by the above test script.

Disabled text field:

Readonly text field:

5.9 Set and assert the value of a hidden field

A hidden field is often used to store a default value.

```
<input type="hidden" name="currency" value="USD"/>
```

The below test script asserts the value of the above hidden field and changes its value using JavaScript.

```
the_hidden_elem = driver.findElement(By.name( "currency"))
driver.findElement(By.name( "currency")).getAttribute("value").then(function(the_value){
    assert.equal("USD", the_value);
});

driver.executeScript("arguments[0].value = 'AUD';", the_hidden_elem);
driver.findElement(By.name( "currency")).getAttribute("value").then(function(the_value){
    assert.equal("AUD", the_value);
});
```

6. Radio button

☒ Male
☐ Female

HTML Source

```
<input type="radio" name="gender" value="male" id="radio_male" checked="true">Male<br>  
<input type="radio" name="gender" value="female" id="radio_female">Female
```

6.1 Select a radio button

```
// driver.get("file://" + __dirname + "/../../site/radio_button.html");  
  
driver.findElement(By.xpath("//input[@name='gender' and @value='female']")).click();  
driver.sleep(200);  
driver.findElement(By.xpath("//input[@name='gender' and @value='male']")).click();
```

The radio buttons in the same radio group have the same name. To click one radio option, the value needs to be specified. Please note that the value is not the text shown next to the radio button, that is the label. To find out the value of a radio button, inspect the HTML source.

As always, if there are IDs, using `id` finder is easier.

```
driver.findElement(By.id("radio_female")).click();
```

6.2 Clear radio option selection

It is OK to click a radio button that is currently selected, however, it would not have any effect.

```
driver.findElement(By.id("radio_female")).click();
// already selected, no effect
driver.findElement(By.id("radio_female")).click();
```

Once a radio button is selected, you cannot just clear the selection in Selenium. (Watir-Classic, another web test framework, can clear radio selection). You need to select another radio button. The test script below will throw an error: “invalid element state: Element must be user-editable in order to clear it.”

```
driver.findElement(By.xpath("//input[@name='gender' and @value='female']")).clear().then(\
function() {
}).catch(function(e) {
  console.log("not allow clear currently selected radio button, select another one");
  driver.findElement(By.xpath("//input[@name='gender' and @value='male']")).click();
});
```

6.3 Assert a radio option is selected

```
let xpathStr = "//input[@name='gender' and @value='female']";
driver.findElement(By.xpath(xpathStr)).click();
assert(driver.findElement(By.xpath(xpathStr)).isSelected());
```

6.4 Iterate radio buttons in a radio group

So far we have been focusing on identifying a specific web controls by using one type of locator `findElement`. We may use another type of locator (I call them plural locators): `findElements`.

```
driver.findElements(By.name("gender")).then(function(radios){
  assert.equal(2, radios.length);
  radios.forEach(function(radio){
    radio.getAttribute("Value").then(function(radio_value) {
      if (radio_value == "female") {
        radio.click();
      }
    });
  });
});
```

Different from `findElement` which returns one matched control, `findElements` return a list of them (also known as an array) back. This can be quite handy especially when controls are hard to locate.

6.5 Click Nth radio button in a group

```
driver.findElements(By.name("gender")).then(function(radios){
    radios[1].click();
});
driver.sleep(200);
let xpathStr = "//input[@name='gender' and @value='female']";
assert(driver.findElement(By.xpath(xpathStr)).isSelected());
```



Once I was testing an online calendar, there were many time-slots, and the HTML for each of these time-slots were exactly the same. I simply identified the time slot by using the index (as above) on one of these 'plural' locators.

6.6 Click radio button by the following label

Some .NET controls generate poor quality HTML fragments like the one below:

```
<div id="q1" class="question">
  <div class="question-answer col-lg-5">
    <div class="yes-no">
      <input id="QuestionViewModels_1__SelectedAnswerId" name="QuestionViewModels[1].SelectedAnswerId" type="radio" value="c225306e-8d8e-45b0-8261-22617d9796b5">
      <label for="QuestionViewModels_1__SelectedAnswerId">Yes</label>
    </div>
    <div class="yes-no">
      <input id="QuestionViewModels_1__SelectedAnswerId" name="QuestionViewModels[1].SelectedAnswerId" type="radio" value="85ff8db7-1c58-47a2-a978-581200fb7098">
      <label for="QuestionViewModels_1__SelectedAnswerId">No</label>
    </div>
  </div>
</div>
```

The `id` attribute of the above two radio buttons are the same, and the `values` are meaningless to human. The only thing can be used to identify a radio button is the text in `label` elements. The solution is to use XPath locator. You might have noticed that `input` (radio button) and `label` are siblings in the HTML DOM tree. We can use this relation to come up a XPath that identifies the label text, then the radio button.

```
let xpathStr = "//div[@id='q1']//label[contains(., 'Yes')]/../input[@type='radio']";
driver.findElement(By.xpath(xpathStr)).click();
```

6.7 Customized Radio buttons - iCheck

There are a number of plugins that customize radio buttons into a more stylish form, like the one below (using iCheck).

Gender: ☒ Male ☐ Female

The iCheck JavaScript transforms the radio button HTML fragment

```
<input type="radio" name="sex" id="q2_1" value="male"> Male
```

to

```
<div class="iradio_square-red" style="position: relative;">
  <input type="radio" name="sex" id="q2_1" value="male" style="..." />
  <ins class="iCheck-helper" style="..." />
</div>
```

Here are test scripts to drive iCheck radio buttons.

```
// Error if trying input element directly: Element is not clickable
// driver.findElement(By.id("q2_1")).click();

driver.findElements(By.className("iradio_square-red")).then(function(checkboxes){
  checkboxes[0].click();
  driver.sleep(200); // add some delays for JavaScript to execute
});

driver.findElements(By.className("iradio_square-red")).then(function(checkboxes){
  checkboxes[1].click();
  driver.sleep(100);
});
// More precise with XPath
driver.findElement(By.xpath("//div[contains(@class, 'iradio_square-red')]/input[@type='radio' and @value='male']/..")).click();
```

7. CheckBox

☐ I have a bike
☒ I have a car

HTML Source

```
<input type="checkbox" name="vehicle_bike" value="on" id="checkbox_bike">I have a bike<br>  
<input type="checkbox" name="vehicle_car" id="checkbox_car">I have a car
```

7.1 Check by name

```
// driver.get("file://" + __dirname + "/../site/checkbox.html");  
driver.findElement(By.name("vehicle_bike")).click();
```

7.2 Check by id

To safely check a checkbox, we shall verify it is not checked first.

```
the_checkbox = driver.findElement(By.id("checkbox_car"));  
the_checkbox.isSelected().then(function(selected) {  
    if (!selected) { the_checkbox.click(); }  
});
```

7.3 Uncheck a checkbox

```
the_checkbox = driver.findElement(By.name("vehicle_bike"));
the_checkbox.click();
the_checkbox.isSelected().then(function(selected) {
    if (selected) { the_checkbox.click(); }
});
```

7.4 Assert a checkbox is checked (or not)

```
the_checkbox = driver.findElement(By.name("vehicle_bike"));
the_checkbox.click();
driver.sleep(100);
assert(the_checkbox.isSelected());

the_checkbox.click().then(function() {
    driver.sleep(100);
    the_checkbox.isSelected().then(function(selected) {
        assert(!selected);
    });
});
```

7.5 Customized Checkboxes - iCheck

There are a number of plugins that customize radio buttons into a more stylish form, like the one below (using iCheck).

☐ Soccer

☒ Basketball

☐ Baseball

The iCheck JavaScript transforms the checkbox HTML fragment

```
<input type="checkbox" name="sports[]" value="Soccer"> Soccer <br/>
```

to

```
<div class="icheckbox_square-red" style="position: relative;">
  <input type="checkbox" name="sports[]" value="Soccer" style="..."/>
  <ins class="iCheck-helper" style="..."/>
</div>
```

Here are test scripts to drive iCheck checkboxes.

```
driver.findElements(By.className("icheckbox_square-red")).then(function(checkboxes){
  checkboxes[0].click();
  driver.sleep(200); // add some delays for JavaScript to execute
});
```

```
driver.findElements(By.className("icheckbox_square-red")).then(function(checkboxes){
  checkboxes[1].click();
  driver.sleep(100);
});
```

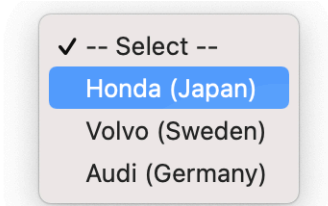
// More precise with XPath

```
driver.findElement(By.xpath("//div[contains(@class, 'icheckbox_square-red')]/input[@type=\n'checkbox' and @value='Soccer']/..")).click();
```


8. Select List

A Select list is also known as a drop-down list or combobox.

Make:



HTML Source

```
<select name="car_make" id="car_make_select">
  <option value="">-- Select --</option>
  <option value="honda">Honda (Japan)</option>
  <option value="volvo">Volvo (Sweden)</option>
  <option value="audi">Audi (Germany)</option>
  <option value="others">Unknown</option>
</select>
```

8.1 Select an option by text

The label of a select list is what we can see in the browser.

```
// driver.get("file://" + __dirname + "/../../site/select_list.html");
driver.findElement(By.name("car_make")).sendKeys("Volvo (Sweden)");
// or starting partial text
driver.findElement(By.name("car_make")).sendKeys("Volvo");
```

The above syntax is quite different from what it is in other four Selenium language bindings, for example, in Java:

```
Select select = new Select(driver.findElement(By.name("car_make_select")));
select.selectByVisibleText("Volvo (Sweden)");
```

I like this syntax better, it is more clear. However, I could not find the equivalent way in Node.js (*the above syntax examples are provided on [the Selenium official documentation page](https://www.selenium.dev/docs/en/support_packages/working_with_select_elements/)¹ for all other languages except JavaScript: “We don’t have a JavaScript code sample yet*

¹https://www.selenium.dev/docs/en/support_packages/working_with_select_elements/

- *Help us out and raise a PR*”), please let me know if you do.

8.2 Select an option by value

The value of a select list is what to be passed to the server.

```
var selElement = driver.findElement(By.name("car_make"));
selElement.findElement(By.css("option[value='others']")).click();
```

8.3 Select an option by iterating all options

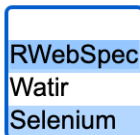
Here I will show you a more complex way to select an option in a select list, not for the sake of complexity, of course. A select list contains options, where each option itself is a valid control in Selenium.

```
let xpathStr = "//select[@id='car_make_select']/option";
driver.findElements(By.xpath(xpathStr)).then(function(the_options) {
  the_options.map(option => {
    option.getText().then(text => {
      if (text == "Volvo (Sweden)")
        option.click();
    });
  });
});
```

8.4 Select multiple options

A select list also supports multiple selections.

Framework:



HTML Source

```
<select id="framework_select" name="test_framework" multiple="multiple">
  <option></option>
  <option value="rwebspec">RSpec</option>
  <option value="watir">Watir</option>
  <option value="selenium">Selenium</option>
</select>
```

Test Script

```
var selElement = driver.findElement(By.name("test_framework"));
selElement.findElement(By.css("option[value='selenium']")).click();
selElement.findElement(By.css("option[value='rwebspec']")).click();
```

8.5 Select options by index

Selecting options by index works for both standard and multiple Select List.

```
// standard
var standardSelElement = driver.findElement(By.name("car_make"));
standardSelElement.findElement(By.css("option:nth-child(2)")).click();

// multiple
var multipleElement = driver.findElement(By.name("test_framework"));
// 1 means the first option, not 0
multipleElement.findElement(By.css("option:nth-child(2)")).click();
multipleElement.findElement(By.css("option:nth-child(4)")).click();
```

8.6 Clear one selection

Clearing selection only applies to multiple select lists.

```

var selElement = driver.findElement(By.name("test_framework"));
selElement.findElement(By.css("option[value='selenium']")).click();
selElement.findElement(By.css("option[value='rwebspec']")).click();

selElement.findElements(By.xpath("option")).then(function(the_options) {
  the_options.map(option => {
    option.isSelected().then(function(selected) {
      if (selected) {
        option.getText().then(text => {
          if (text == "Selenium")
            option.click();
        });
      }
    });
  });
});

```

8.7 Clear all selections

```

var selElement = driver.findElement(By.name("test_framework"));
selElement.findElement(By.css("option[value='selenium']")).click();
selElement.findElement(By.css("option[value='rwebspec']")).click();

selElement.findElements(By.xpath("option")).then(function(the_options) {
  the_options.map(option => {
    option.isSelected().then(function(selected) {
      if (selected) {
        option.click();
      }
    });
  });
});

```

8.8 Assert a label or value in a select list

To verify a label or value is in a select list.

```

my_select = driver.find_element(:id, "car_make_select")
select_texts = my_select.find_elements( :tag_name => "option" ).collect{|option|
  option.text }
expect(select_texts.include?("Audi (Germany)")).to be_truthy

select_values = my_select.find_elements( :tag_name => "option" ).collect{|option|
  option["value"] }
expect(select_values.include?("audi")).to be_truthy

```

8.9 Assert selected option label

To verify a particular option text is currently selected in a select list:

```

test.it("Assert option (label or value) in a select list", async function() {
  var selElem = driver.findElement(By.id("car_make_select"));
  const selectedOptionTexts = [];
  const selectedOptionValues = [];
  await selElem.findElements(By.xpath("option")).then(function(the_options) {
    the_options.map(option => {
      option.getText().then(text => {
        selectedOptionTexts.push(text)
      });
      option.getAttribute("value").then(val => {
        selectedOptionValues.push(val)
      });
    });
  });

  console.log(selectedOptionTexts);
  console.log(selectedOptionValues);
  assert(selectedOptionTexts.includes('Audi (Germany)'));
  assert(selectedOptionValues.includes('honda'));
});

```

8.10 Assert the value of a select list

Another quick (and simpler) way to check the current selected value of a select list:

```
var selElem = driver.findElement(By.name("car_make"))
selElem.sendKeys("Honda");
selElem.getAttribute('value').then(function(selected) {
    assert.equal("honda", selected)
});
```

8.11 Assert multiple selections

A multiple select list can have multiple options being selected.

```
test.it ("Assert multiple selected options", async function() {
    var selElement = driver.findElement(By.name("test_framework"));
    selElement.findElement(By.css("option[value='selenium']")).click();
    selElement.findElement(By.css("option[value='rwebspec']")).click();

    const selectedOptionTexts = [];
    await selElement.findElements(By.xpath("option")).then(function(the_options) {
        the_options.map(option => {
            option.isSelected().then(function(selected) {
                if (selected) {
                    option.getText().then(text => {
                        selectedOptionTexts.push(text);
                    });
                }
            });
        });
    });

    console.log(selectedOptionTexts);
    assert.equal(2, selectedOptionTexts.length);
    assert.equal("RWebSpec", selectedOptionTexts[0]); // The order is based the option list
    assert.equal("Selenium", selectedOptionTexts[1]);
});
```

Please note, even though the test script selected 'Selenium' first, when it comes to assertion, the first selected option is 'RWebSpec', not 'Selenium'.

9. Navigation and Browser

Driving common web controls were covered from chapters 2 to 7. In this chapter, I will show how to manage browser windows and page navigation in them.

9.1 Go to a URL

```
driver.get("http://testwisely.com");
```

9.2 Visit pages within a site

`driver.get()` takes a full URL. Most of time, testers test against a single site and specifying a full URL (such as `http://...`) is not necessary. We can create a reusable function to simplify its usage.

```
var site_root_url = "http://test.testwisely.com";

// ...

function visit(path) {
    driver.get(site_root_url + path);
}

// ...

test.it("Go to page within the site using function", function() {
    visit("/demo");
    visit("/demo/survey");
    visit("/"); // home page
});
```

Apart from being more readable, there is another benefit with this approach. If you want to run the same test against at a different server (the same application deployed on another machine), we only need to make one change: the value of `site_root_url`.

```
site_root_url = "http://dev.testwisely.com";
```

9.3 Perform actions from right click context menu such as 'Back', 'Forward' or 'Refresh'

Operations with right click context menu are commonly page navigations, such as “Back to previous page”. We can achieve the same by calling the test framework’s navigation operations directly.

```
driver.navigate().back();  
driver.navigate().refresh();  
driver.navigate().forward();
```

9.4 Open browser in certain size

Many modern web sites use responsive web design, that is, page content layout changes depending on the browser window size. Yes, this increases testing effort, which means testers need to test web sites in different browser window sizes. Fortunately, Selenium has a convenient way to resize the browser window.

```
driver.manage().window().setSize(1024, 768);
```

9.5 Maximize browser window

```
driver.manage().window().maximize();  
driver.sleep(300); // add delay to see the effect  
driver.manage().window().setSize(1024, 768);
```

9.6 Move browser window

We can move the browser window (started by the test script) to a certain position on screen, (0, 0) is the top left of the screen. The position of the browser’s window won’t affect the test results. This might be useful for utility applications, for example, a background video program can capture a certain area on screen.


```
driver.manage().window().setPosition(100, 200);  
driver.sleep(300);  
driver.manage().window().setPosition(0, 0);
```

9.7 Minimize browser window

Surprisingly, there is no minimize window function in Selenium. The hack below achieves the same:

```
driver.manage().window().setPosition(-2000, 0); // hide  
driver.findElement(By.linkText("Hyperlink")).click(); // still can use  
driver.sleep(300);  
driver.manage().window().setPosition(0, 0);
```

While the browser's window is minimized, the test execution still can run.

9.8 Scroll focus to control

For certain controls are not viewable in a web page (due to JavaScript), WebDriver is unable to click on them by returning an error like *"Element is not clickable at point (1180, 43)"*. The solution is to scroll the browser view to the control.

```
driver.manage().window().setSize(1024, 208); // make browser small  
elem = driver.findElement(By.name("submit_action_2"));  
elem_pos = elem.getLocation().y  
driver.executeScript("window.scroll(0, " + elem_pos + ");"); // scroll  
driver.sleep(300)  
elem.click();  
driver.manage().window().setSize(1024, 768);
```

9.9 Switch between browser windows or tabs

A `target='_blank'` hyperlink opens a page in another browser window or tab (depending on the browser setting). Selenium drives the browser within a scope of one browser window. However, we can use Selenium's `switchTo` function to change the target browser

```
// click a target='_blank' link
driver.findElement(By.linkText("Open new window")).click();
driver.getAllWindowHandles().then(function(allWindows){
    // switch to the last tab
    driver.switchTo().window(allWindows[allWindows.length - 1])
});

driver.findElement(By.tagName("body")).getText().then(function(page_text) {
    assert(page_text.contains("This is url link page"));
});

driver.getAllWindowHandles().then(function(allWindows){
    driver.switchTo().window(allWindows[0]) // back to first tab
    assert(driver.findElement(By.linkText("Open new window")).isDisplayed());
});
```

9.10 Remember current web page URL, then come back to it later

We can store the page's URL into an instance variable (url, for example).

```
var url; // instance variable

test.beforeEach(function() {
    // runs before each test in this block
    url = driver.getCurrentUrl();
});

//...

test.it("Go back to a URL set outside this test case", function() {
    driver.findElement(By.linkText("Button")).click();
    // ...
    driver.get(url);
});
```

In previous recipes, I used local variables to remember some value, and use it later. A local variable only works in its local scope, typically within one test case (in our context, or more specifically between `test.it("...", function() { to } }`).

In this example, the `url` variable is used in `test.beforeEach` scope. To make it accessible to the test cases in the test script file, I define it as an instance variable `url`.

10. Assertion

Without assertions (or often known as checks), a test script is incomplete. Common assertions for testing web applications are:

- page title (equals)
- page text (contains or does not contain)
- page source (contains or does not contain)
- input element value (equals)
- display element text (equals)
- element state (selected, disabled, displayed)

10.1 Assert page title

```
driver.getTitle().then(function(the_page_title) {  
    assert.equal("Assertion Test Page", the_page_title);  
});
```

10.2 Assert Page Text

Example web page

```
Text assertion with a  (tab before), and  
(new line before)!
```

HTML source

```
<PRE>Text assertion with a  (<b>tab</b> before), and  
(new line before)!</PRE>
```

We can use `string.indexOf()` to check whether a string contains another string. It works, but not easy to read.

```
String.prototype.contains = function(it) { return this.indexOf(it) != -1; };
```

Test script

```
driver.findElement(By.tagName("body")).getText().then(function(the_page_text) {  
    assert(the_page_text.includes(matching_str))  
});
```

Please note the `findElement(By.tagName("body")).getText()` returns the text view of a web page after stripping off the HTML tags, but may not be exactly the same as we saw on the browser.

10.3 Assert Page Source

The page source is raw HTML returned from the server.

```
html_fragment = "Text assertion with a (<b>tab</b> before), and \n(new line before)!"  
driver.getPageSource().then(function(the_page_source) {  
    assert(the_page_source.contains(html_fragment))  
});
```

10.4 Assert Label Text

HTML source

```
<label id="receipt_number">NB123454</label>
```

Label tags are commonly used in web pages to wrap some text. It can be quite useful to assert a specific text.

```
driver.findElement(By.id("label_1")).getText().then(function(the_elem_text) {  
    assert.equal("First Label", the_elem_text);  
});
```

10.5 Assert Span text

HTML source

```
<span id="span_2">Second Span</span>
```

From testing perspectives, spans are the same as labels, just with a different tag name.

```
driver.findElement(By.id("span_2")).getText().then(function(the_elem_text) {  
    assert.equal("Second Span", the_elem_text);  
});
```

10.6 Assert Div text or HTML

Example page

```
Wise Products  
TestWise  
BuildWise
```

HTML source

```
<div id="div_parent">  
    Wise Products  
    <div id="div_child_1">  
        TestWise  
    </div>  
    <div id="div_child_2">  
        BuildWise  
    </div>  
</div>
```

Test script

```
// text  
driver.findElement(By.id("div_child_1")).getText().then(function(elem_text) {  
    assert.equal("TestWise", elem_text);  
});  
  
driver.findElement(By.id("div_parent")).getText().then(function(elem_text) {  
    assert.equal("Wise Products\nTestWise\nBuildWise", elem_text);  
});  
  
// HTML  
the_element = driver.findElement(By.id("div_parent"));
```

```
driver.executeScript("return arguments[0].outerHTML;", the_element).then(function(html) {
    assert.equal("<div id=\"div_parent\">\n                Wise Products\n<div id=\"div_child_1\">\n                TestWise\n                </div>\n<div id=\"div_child_2\">\n                BuildWise\n                </div>\n</div>", html)
});
```

10.7 Assert Table text

HTML tables are commonly used for displaying grid data on web pages.

Example page

A	B
a	b

HTML source

```
<table id="aha_table" cellpadding="1" border="1" width="30%">
  <tr id="row_1">
    <td id="cell_1_1">A</td>
    <td id="cell_1_2">B</td>
  </tr>
  <tr id="row_2">
    <td id="cell_2_1">a</td>
    <td id="cell_2_2">b</td>
  </tr>
</table>
```

Test script

```
driver.findElement(By.id("alpha_table")).getText().then(function(the_elem_text){
    assert.equal("A B\na b", the_elem_text);
});
```

```
driver.findElement(By.id("alpha_table")).then(function(the_element) {
    driver.executeScript("return arguments[0].outerHTML;", the_element).then(
        function(the_elem_html) {
            assert(the_element_html.contains("<td id=\"cell_1_1\">A</td>"))
        }
    );
});
```

10.8 Assert text in a table cell

If a table cell (td tag) has a unique ID, it is easy.

```
driver.findElement(By.id("cell_1_1")).getText().then(function(cell_text) {  
    assert.equal("A", cell_text);  
});
```

An alternative approach is to identify a table cell using row and column indexes (both starting with 0).

```
driver.findElement(By.xpath("//table/tbody/tr[2]/td[2]")).getText().then(  
    function(the_elem_text) {  
        assert.equal("b", the_elem_text);  
    });
```

10.9 Assert text in a table row

```
driver.findElement(By.id("row_1")).getText().then(function(the_elem_text) {  
    assert.equal("A B", the_elem_text);  
});
```

10.10 Assert image present

```
assert(driver.findElement(By.id("next_go")).isDisplayed());
```

10.11 Assert element location and width

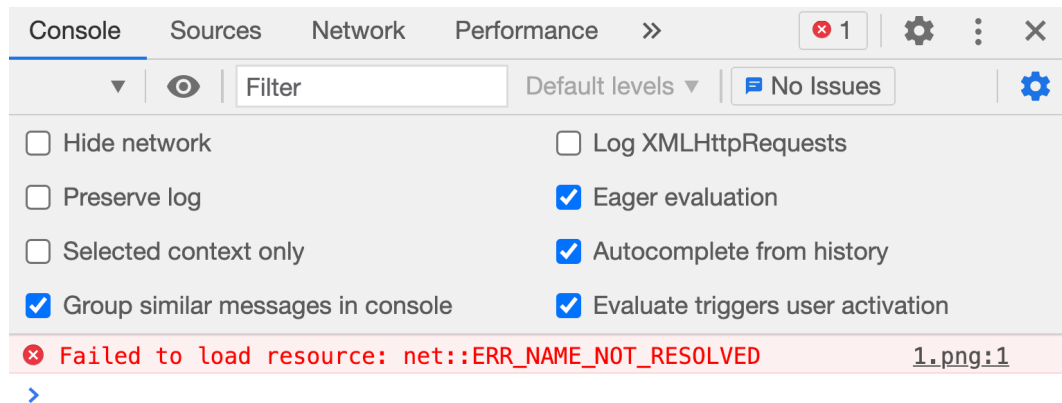
```
driver.findElement(By.id("next_go")).getSize().then(function(the_size) {
    assert.equal(46,the_size.width)
});
driver.findElement(By.id("next_go")).getLocation().then(function(the_loc) {
    assert(the_loc.x > 100)
});
```

10.12 Assert element CSS style

```
driver.findElement(By.id("highlighted")).getCssValue("background-color").then(
    function(bc) {
        assert.equal("rgba(206, 218, 227, 1)", bc)
    });
driver.findElement(By.id("highlighted")).getCssValue("font-size").then(function(fsize) {
    assert.equal("15px", fsize)
});
```

10.13 Assert JavaScript errors on a web page

JavaScripts errors can be inspected in browser's console, as below:



Here is how to detect JavaScript errors with Selenium WebDriver.


```
driver.get("http://testwisely.com/demo/customer-interview");
driver.sleep(300)
driver.manage().logs().get("browser").then(function(browser_log) {
    console.log(browser_log);
    assert.equal(1, browser_log.length);
    console.log(JSON.stringify(browser_log[0]));
    assert(JSON.stringify(browser_log[0]).contains("net::ERR_NAME_NOT_RESOLVED"));
});
```

Verify no JS errors.

```
driver.get("http://testwisely.com/demo");
driver.sleep(300)
driver.manage().logs().get("browser").then(function(browser_log) {
    assert.equal(0, browser_log.length);
});
```

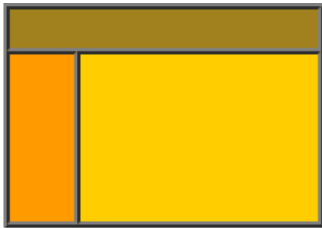
Please note this feature is browser dependent, it works on Chrome.

11. Frames

HTML Frames are treated as independent pages, which is not a good web design practice. As a result, few new sites use frames nowadays. However, there a quite a number of sites that uses iframes.

11.1 Testing Frames

Here is a layout of a fairly common frame setup: navigations on the top, menus on the left and the main content on the right.



HTML Source

```
<frameset rows="100,*" frameborder="0" border="0" framespacing="0">
  <frame name="topNav" src="top_nav.html">
  <frameset cols="200,*" frameborder="0" border="0" framespacing="0">
    <frame name="menu" id="menu_frame" src="menu_1.html" marginheight="0"
      marginwidth="0" scrolling="auto" noresize>
    <frame name="content" src="content.html" marginheight="0" marginwidth="0"
      scrolling="auto" noresize>
  </frameset>
</frameset>
```

To test a frame with Selenium, we need to identify the frame first by ID or NAME, and then switch the focus on it. The test steps after will be executed in the context of selected frame. Use `switch_to.default_content()` to get back to the page (which contains frames).

```
driver.switchTo().frame("topNav") // name
driver.findElement(By.linkText("Menu 2 in top frame")).click();

// need to switch to default before another switch
driver.switchTo().defaultContent()
driver.sleep(200);
// the below seems not working for Chrome, fine on firefox
driver.switchTo().frame("menu") // using attr: name
// or driver.switchTo().frame("menu_frame") // using attr: id
driver.findElement(By.linkText("Green Page")).click();

driver.switchTo().defaultContent()
driver.switchTo().frame("content")
driver.findElement(By.linkText("Back to original page")).click();
```

This script clicks a link in each of three frames: top, left menu and content.

11.2 Testing iframe

An iframe (Inline Frame) is an HTML document embedded inside another HTML document on a web site.

Example page

Enter name:

Username:

Password:

HTML Source

```
<IFRAME frameborder='1' id="Frame1" src="login_iframe.html"
  style="HEIGHT: 100px; WIDTH: 320px; MARGIN=0" SCROLLING="no" >
</IFRAME>
```

The test script below enters text in the main page, fills the sign in form in an iframe, and ticks the checkbox on the main page:

```
driver.get(site_root_url + "/iframe.html")
driver.findElement(By.name("user")).sendKeys("agileway")
driver.switchTo().frame("Frame1")
driver.findElement(By.name("username")).sendKeys("tester")
driver.findElement(By.name("password")).sendKeys("TestWise")
driver.findElement(By.id("loginBtn")).click();
driver.sleep(200);
driver.getPageSource().then(function(the_page_source) {
    assert(the_page_source.contains("Signed in"))
});
driver.switchTo().defaultContent();
driver.findElement(By.id("accept_terms")).click();
```

The web page after test execution looks as below:

Enter name:



☒ I accept terms and conditions

Please note that the content of the iframe changed, but not the main page.

11.3 Test multiple iframes

A web page may contain multiple iframes.

```
driver.get(site_root_url + "/iframes.html")
driver.switchTo().frame(0)
driver.findElement(By.name("username")).sendKeys("agileway")
driver.switchTo().defaultContent()
driver.switchTo().frame(1)
driver.findElement(By.id("radio_male")).click();
```


12. Testing AJAX

AJAX (an acronym for Asynchronous JavaScript and XML) is widely used in web sites nowadays (Gmail uses AJAX a lot). Let's look at an example first:

NetBank

To Account:

Enter Amount:



On clicking 'Transfer' button, an animated loading image showed up indicating 'transfer in progress'.

NetBank

To Account:

Enter Amount:

Receipt No: 6671

Receipt Date: 04/03/2021

After the server processing the request, the loading image is gone and a receipt number is displayed.

From testing perspective, a test step (like clicking 'Transfer' button) is completed immediately. However the updates to parts of a web page may happen after unknown delay, which differs from traditional web requests.

There are 2 common ways to test AJAX operations: waiting enough time or checking the web page periodically for a maximum given time.

12.1 Wait within a time frame

After triggering an AJAX operation (clicking a link or button, for example), we can set a timer in our test script to wait for all the asynchronous updates to occur before executing next step.

```
test.it("Wait enough long time (using sleep)", function() {  
    this.timeout(15000);  
    // ..  
    driver.findElement(By.xpath("//input[@value='Transfer']")).click();  
    driver.sleep(10000)  
    driver.findElement(By.tagName("body")).getText().then(function(text){  
        assert(text.contains("Receipt No:"))  
    });  
});
```

`driver.sleep(10000)` means waiting for 10 seconds, after clicking ‘Transfer’ button. 10 seconds later, the test script will check for the ‘Receipt No:’ text on the page. If the text is present, the test passes; otherwise, the test fails. In other words, if the server finishes the processing and return the results correctly in 11 seconds, this test execution would be marked as ‘failed’.

12.2 Explicit Waits until Time out

Apparently, the waiting for a specified time is not ideal. If the operation finishes earlier, the test execution would still be on halt. Instead of passively waiting, we can write test scripts to define a wait statement for certain condition to be satisfied until the wait reaches its timeout period. If Selenium can find the element before the defined timeout value, the code execution will continue to next line of code.

```
test.it("Wait enough long time (using Explicit Waits)", function() {
    this.timeout(15000);
    driver.findElement(By.name("account")).sendKeys("Cheque");
    driver.findElement(By.id("rcptAmount")).sendKeys("250");
    driver.findElement(By.xpath("//input[@value='Transfer']")).click();
    driver.wait(function () {
        return driver.isElementPresent(By.id("receiptNo"));
    }, 10000);
});
```

Besides `isElementPresent`, there are many others such as `elementToBeClickable`, `presenceOfAllElementsLocatedBy`, ..., etc. The full list can be found on [this JavaDoc page](http://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/support/ui/ExpectedConditions.html)¹.

12.3 Implicit Waits until Time out

An implicit wait is to tell Selenium to poll finding a web element (or elements) for a certain amount of time if they are not immediately available. The default setting is 0. Once set, the implicit wait is set for the life of the `WebDriver` object instance, until its next set.

```
driver.manage().timeouts().implicitlyWait(11000);
driver.findElement(By.xpath("//input[@value='Transfer']")).click();
assert(driver.isElementPresent(By.id("receiptNo")));
driver.manage().timeouts().implicitlyWait(0);
```

12.4 Wait AJAX Call to complete using JQuery

If the target application uses JQuery for Ajax requests (most do), you may use a JavaScript call to check active Ajax requests: `jQuery.active` is a variable JQuery uses internally to track the number of simultaneous AJAX requests.

1. drive the control to initiate AJAX call
2. wait until the value of `jQuery.active` is zero
3. continue the next operation

The *waiting* is typically implemented in a reusable function.

¹<http://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/support/ui/ExpectedConditions.html>

```
function waitForAjaxComplete(maxSeconds) {
  var is_ajax_comlete = false;
  for (i = 1; i <= maxSeconds; i++) {
    driver.executeScript("return jQuery.active == 0;").then(
      function(is_ajax_complete) {
        console.log("is_ => " + is_ajax_comlete)
        if (is_ajax_complete) {
          return;
        }
      });
    driver.sleep(1000);
  }
  // throw "Timed out waiting for AJAX call after " + maxSeconds + " seconds";
}

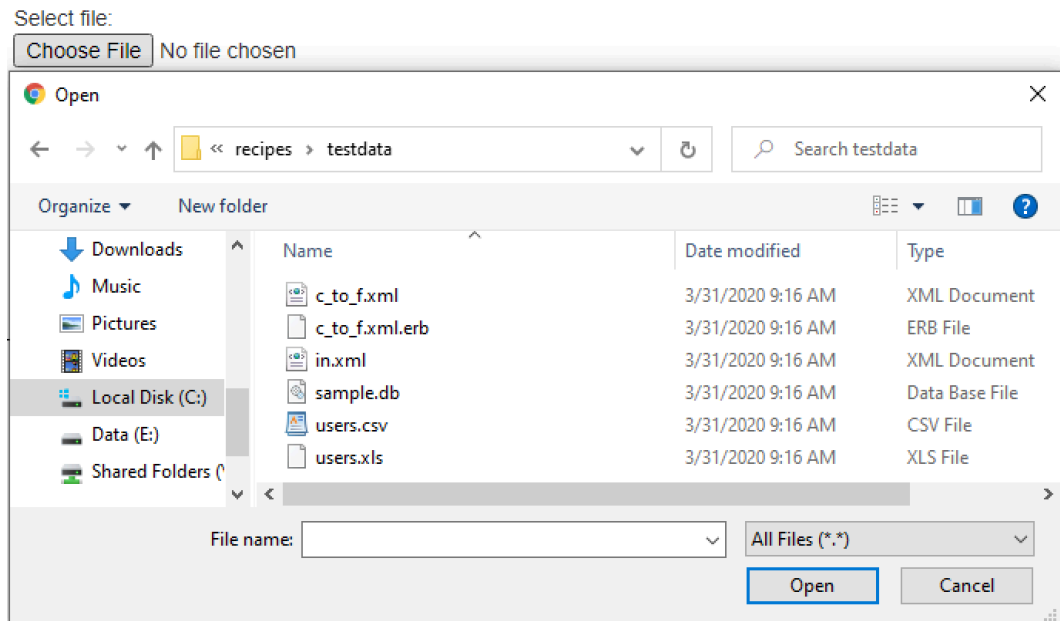
test.it("Wait AJAX complete using JQuery Active flag", function() {
  this.timeout(timeOut);
  driver.findElement(By.name("account")).sendKeys("Cheque");
  driver.findElement(By.id("rcptAmount")).sendKeys("250");
  driver.findElement(By.xpath("//input[@value='Transfer']")).click().then(function(){
    waitForAjaxComplete(10)
    assert(driver.isElementPresent(By.id("receiptNo")) );
  });
});
```


13. File Upload and Popup dialogs

In this chapter, I will show you how to handle file upload and popup dialogs. Most of pop up dialogs, such as 'Choose File to upload', are native windows rather than browser windows. This would be a challenge for testing as Selenium only drives browsers. If one pop up window is not handled properly, test execution will be on halt.

13.1 File upload

Example page



HTML Source

```
<input type="file" name="document[file]" id="files" size="60"/>
```

Test script

```
driver.get("file://" + __dirname + "../../../site/popup.html");  
  
var filePath = "C:\\testdata\\logo.png";  
driver.findElement(By.name("document[file]")).sendKeys(filePath);
```

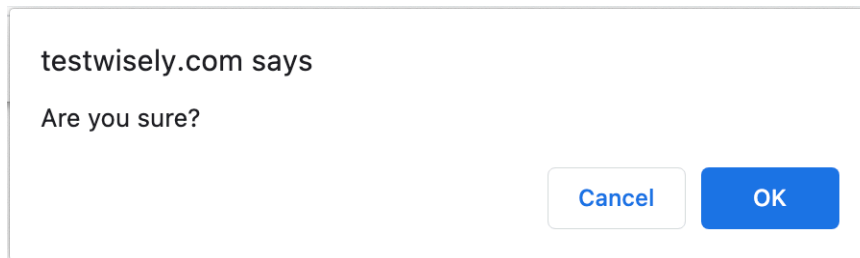
The first slash of \\ (for Windows platform) is for escaping the later one, the whole purpose is to pass the value “C:\testdata\logo.png” to the control.

Some might say, hard coding a file path is not a good practice. It’s right, it is generally better to include your test data files within your test project, then use relative paths to refer to them, as the example below:

```
var filePath = __dirname + "../testdata/users.csv";  
// console.log("file: " + filePath);  
driver.findElement(By.name("document[file]")).sendKeys(filePath);
```

13.2 JavaScript pop ups

JavaScript pop ups are created using javascript, commonly used for confirmation or alerting users.



There are many discussions on handling JavaScript Pop ups in forums and Wikis. I tried several approaches. Here I list two stable ones:

Handle JavaScript pop ups using Alert API

```

driver.get("http://testwisely.com/demo/popups");
driver.findElement(By.xpath("//input[contains(@value, 'Buy Now')]")).click();
driver.switchTo().alert().then(function(the_alert) {
  the_alert.getText().then(function(alert_text){
    console.log(alert_text);
    if (alert_text == "Are you sure?") {
      the_alert.accept();
    } else {
      the_alert.dismiss();
    }
  });
});

```

Handle JavaScript pop ups with JavaScript

```

driver.executeScript("window.confirm = function() { return true; }");
driver.executeScript("window.alert = function() { return true; }");
driver.executeScript("window.prompt = function() { return true; }");
driver.findElement(By.xpath("//input[contains(@value, 'Buy Now')]")).click();

```

Different from the previous approach, the pop up dialog is not even shown.

This recipe is courtesy of [Alister Scott's WatirMelon blog](#)¹

13.3 Timeout on a test

When a pop up window is not handled, it blocks the test execution. This is worse than a test failure when running a number of test cases. For operations that are prone to hold ups, we can add a time out with a specified maximum time.

```

test.it("Test timeout", function() {
  this.timeout(5000);
  // operations here
}

```

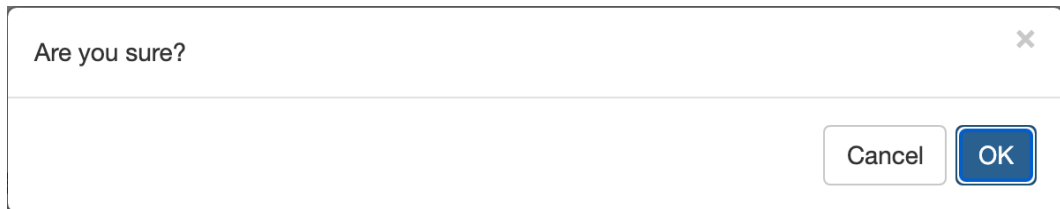
The below is a sample test output when a test case execution times out.

¹<http://watirmelon.com/2010/10/31/dismissing-pesky-javascript-dialogs-with-watir/>

```
java.lang.Exception: test timed out after 5000 milliseconds
```

13.4 Modal style dialogs

Flexible Javascript libraries, such as [Bootstrap Modals](http://getbootstrap.com/javascript/#modals)², replace the default JavaScript alert dialogs used in modern web sites. Strictly speaking, a modal dialog like the one below is not a pop-up.



Comparing to the raw JS *alert*, writing automated tests against modal popups is easier.

```
driver.findElement(By.id("bootbox_popup")).click();  
driver.sleep(200);  
driver.findElement(By.xpath("//div[@class='modal-footer']/button[text()='OK']")).click();
```

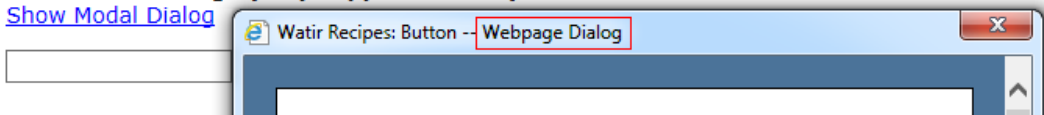
13.5 Internet Explorer modal dialog

Modal dialog, only supported in Internet Explorer, is a dialog (with 'Webpage dialog' suffix in title) that user has to deal with before interacting with the main web page. It is considered as a bad practice, and it is rarely found in modern web sites. However, some unfortunate testers might have to deal with modal dialogs.

Example page

Modal Web Dialogs (only supported in IE)

[Show Modal Dialog](#)



HTML Source

²<http://getbootstrap.com/javascript/#modals>

```
<a href="javascript:void(0);" onclick="window.showModalDialog('button.html')">  
  Show Modal Dialog</a>
```

Test script

```
driver.findElement(By.linkText("Show Modal Dialog")).click();  
driver.getAllWindowHandles().then(function(allWindows){  
  // switch to the modal win  
  driver.switchTo().window(allWindows[allWindows.length - 1]);  
  driver.findElement(By.name("user")).sendKeys("in_modal");  
});  
  
driver.getAllWindowHandles().then(function(allWindows){  
  driver.switchTo().window(allWindows[0]) // back to the main  
  driver.findElement(By.name("status")).sendKeys("done");  
});
```

14. Debugging Test Scripts

Debugging usually means analyzing and removing bugs in the code. In the context of automated functional testing, debugging is to find out why a test step did not execute as expected and fix it.

14.1 Print text for debugging

```
driver.get("file://" + __dirname + "../../../site/assert.html");
driver.getTitle().then(function(page_title){
  console.log("Now on page: " + page_title);
});
driver.findElement(By.id("app_id")).getText().then(function(appNo){
  console.log("Application number is " + appNo);
});
```

Here is the output from executing the above test from command line:

```
Now on page: Assertion Test Page
Application number is 1234
```

For complex object, its output in default string format might not be meaningful. We can convert dump it in JSON format:

```
console.log(JSON.stringify(complexObject))
```

When the test is executed in a Continuous Integration server, output is normally captured and shown. This can be quite helpful on debugging test execution.

14.2 Write page source or element HTML into a file

When the text you want to inspect is large (such as the page source), printing out the text to a console will not be helpful (too much text). A better approach is to write the output to a temporary file and inspect it later. It is often a better way to write to a temporary file, and use some other tool to inspect later.

```
var output_file = __dirname + "/tmp/login_page.html";
var fs = require('fs');
// console.log(output_file);
driver.getPageSource().then(function(page_source){
  fs.writeFile(output_file, page_source, function(err) {
    if (err) {
      return console.log(err)
    }
    console.log("Saved to file: " + output_file)
  });
});
```

You can also just dump a specific part of web page:

```
var elem = driver.findElement(By.id("div_parent"));
driver.executeScript("return arguments[0].outerHTML;", elem).then(function(elem_html) {
  fs.writeFile(__dirname + "/tmp/login_parent.xhtml", elem_html);
});
```

14.3 Take screenshot

Taking a screenshot of the current browser window when an error/failure happened is a good debugging technique. Selenium supports it in a very easy way.

```
var fs = require('fs');
driver.takeScreenshot().then(function(data) {
  // Base64 encoded png
  fs.writeFileSync(__dirname + "/tmp/screenshot1.png", data, 'base64');
});
```

The above works. However, when it is run the second time, it will return error “The file already exists”. A simple workaround is to write a file with timestamped file name, as below:

```
// save to timestamped file, e.g. screenshot-12070944.png
Date.prototype.yyyymmdd = function() {
  var mm = this.getMonth() + 1; // getMonth() is zero-based
  var dd = this.getDate();

  return [this.getFullYear(), !mm[1] && '0', mm, !dd[1] && '0', dd].join('');
};

function writeScreenshot(data) {
  name = new Date().yyymmdd() + ".png";
  var screenshotPath = __dirname + "/tmp/";
  var fs = require('fs');
  fs.writeFileSync(screenshotPath + name, data, 'base64');
};

test.it("Take screenshot saved to timestamped file", function() {
  driver.takeScreenshot().then(function(data) {
    writeScreenshot(data);
  });
});
```

14.4 Run single test case with Mocha

When we run Mocha tests (`mocha script_spec.js`), it will run all test cases in the test script file. It is fine for regression testing. During development, we often just want to run one specific test case in a test script file. To run a single test in Mocha, add `.only` to the test case like below.

```
test.it("Another test case", function() {
  // ...
});

test.it.only("Just run this test case", function() {
  // ...
});
```

Then `mocha script_spec.js` will only run the second test case.

14.5 Run tests matching a pattern with Mocha

Mocha can also the tests (by name) matching a patten. For example, there are 4 tests in this test script file.

```
test.it("Print out text", function() {  
  // ...  
}  
test.it("Take screenshot", function() {  
  // ...  
}  
  
test.it("Write page or element html to file ", function() {  
  // ...  
}  
  
test.it("Take screenshot saved to timestamped file", function() {  
  // ...  
}
```

By supplying options `-g <pattern>`,

```
> mocha -g screenshot ch14_debugging_spec.js
```

Mocha will only run the two tests containing ‘screenshot’:

```
✓ Take screenshot (1230ms)  
✓ Take screenshot saved to timestamped file (126ms)
```

For more Mocha running options, visit <https://mochajs.org/>¹.

14.6 Leave browser open after test finishes

Once an error or failure occurred during test execution, a tester’s immediate instinct is to check two things:

1. which test statement failed?

The first one can be easily found in command line output, identified by the line numbers in the backtrace. Some IDEs even highlights the error line.

¹<https://mochajs.org/>

2. the page content in the browser (for manual inspection).

We need the browser to stay open to see the web page after the execution of one test case completes. However, we don't want that when running a group of tests, as it will affect the execution of the following test cases.

Usually, we put browser closing statements in `test.After` hook like the below:

```
test.after(function() {
  // driver.quit();
});
```

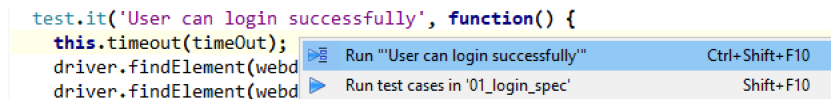
Ideally, we would like to keep the browser open when running an individual test case and close the browser when running multiple test script files, without the need to toggle the 'quit' test step. [TestWise²](http://testwisely.com/testwise), a testing IDE for Selenium, has this feature. It may be a possible extension you can add to your JavaScript IDE. What I can tell you is that this feature is quite useful.

14.7 Run selected test steps against current browser

In my early days with Selenium WebDriver, One feature I missed most from Watir was the ability to attach execution to an existing browser (IE) window. This is an extremely useful feature for debugging tests. When a test step fails, after analyzing and/or modifying (the test scripts or application), we often would like to rerun the test case continuing from where it stopped, rather than starting from the beginning again.

I needed this feature so much that I created a testing tool (TestWise) to achieve it. That is, creating a special test script file with selected test steps, executions of this special test will be **attached** to the last browser. Let me illustrate it.

1. Run an individual test



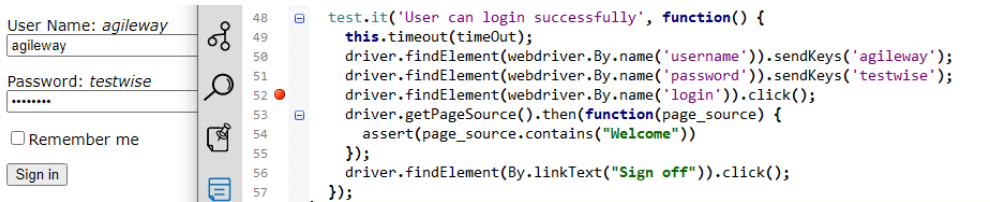
```
test.it('User can login successfully', function() {
  this.timeout(timeout);
  driver.findElement(webd
  driver.findElement(webd
```

The screenshot shows a code editor with a test script. A right-click context menu is open over the code, displaying two options: "Run 'User can login successfully'" with a keyboard shortcut of "Ctrl+Shift+F10", and "Run test cases in '01_login_spec'" with a keyboard shortcut of "Shift+F10".

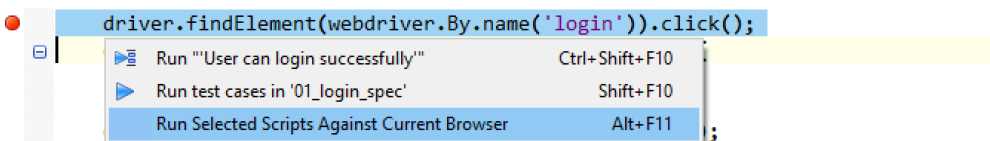
In this execution mode, TestWise will keep the browser open after the test execution completes.

²<http://testwisely.com/testwise>

2. Encounter a test failure

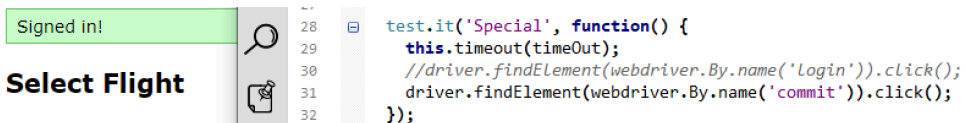


3. Select one or many test steps, and select 'Run Selected Scripts Against Current Browser' in the right-click context menu.



TestWise will create a new debugging test script file (*debugging_spec.js*) with the selected test steps, and run it. The same test failure is expected. The execution, however, is different this time. It reuses the last browser and runs only the selected steps.

4. Edit and run the test steps in the *debugging_spec.js* directly against the browser, as many times as necessary.



This special test script look like this:

```
test.describe('Debugging', function() {

  test.before(function() {
    this.timeout(timeOut);
    driver = new webdriver.Builder().forBrowser(helper.browserType()).
      setChromeOptions(helper.debuggingChromeOptions()).build();
  });

  test.it('Special', function() {
    this.timeout(timeOut);
```

```
// edit your test scripts here to attach to the last open browser
driver.findElement(webdriver.By.name('commit')).click();
});

});
```

This special test only ‘attaches’ to the Chrome browser window of the last execution from TestWise. I did not research whether it works for other browser types, for me, being able to debug in the most popular browser is good enough. The completed (selenium) test scripts, after the debugging is done, of course can be run against any browser.

The mechanism is using Chrome’s remote debugging port. You may implement this as an extension to your choice of testing tools.

15. Test Data

Gathering test data is an important but often neglected activity. With the power of Java programming, testers now have a new ability to prepare test data.

15.1 Get date dynamically

```
// assume today is 2016-07-18
var today = new Date();
// generate today's time as string, e.g. 18-7-2016 8:14
var dateStr = today.getDate() + "-" + (today.getMonth()+1) + "-" + today.getFullYear() + \
    " " +
    today.getHours() + ":" + today.getMinutes();
driver.findElement(By.name("username")).sendKeys(dateStr);
```

Please note the month is 0-based. As you can see, it seems quite complex using JavaScript's built-in date functions. A much easier and flexible way is to use [Moment.js](http://momentjs.com/)¹.

```
// npm install --save moment
var moment = require('moment');

// current time
console.log(moment().format()); // 2016-07-18T07:53:42+10:00
console.log(moment().format("MMM DD, YYYY")); // Jul 18, 2016
console.log(moment().format("DD/MM/YY")); // 18/07/16

// yesterday and tomorrow
console.log(moment().subtract(1, 'days').format("YYYY-MM-DD")); // 2016-07-17
console.log(moment().add(1, 'days').format("YYYY-MM-DD")); // 2016-07-19

// in fortnight
console.log(moment().add(2, 'weeks').format("YYYY-MM-DD")); // 2016-08-01
```

Check [Moment.js Docs](http://momentjs.com/docs/)² for more usage.

¹<http://momentjs.com/>

²<http://momentjs.com/docs/>

15.2 Get a random boolean value

A boolean value means either *true* or *false*. Getting a random true or false might not sound that interesting. That was what I thought when I first learned it. Later, I realized that it is actually very powerful, because I can fill the computer program (test script as well) with nondeterministic data.

```
function getRandomBoolean() {  
    return Math.random() >= 0.5;  
}
```

For example, in a user sign up form, we could write two cases: one for male and one for female. With random boolean, I could achieve the same with just one test case. If the test case get run many times, it will cover both scenarios.

```
driver.get("file://" + __dirname + "/../../site/radio_button.html");  
var randomGender = getRandomBoolean() ? "male" : "female";  
driver.findElement(By.xpath("//input[@type='radio' and @name='gender' and @value='" +  
    randomGender + "']")).click();
```

15.3 Generate a number within a range

```
// a number within a range  
function getRandomInteger(min, max) {  
    return Math.floor(Math.random() * (max - min + 1) + min);  
}
```

The test statement below will enter a number between 16 to 99. If the test gets run hundreds of times, not a problem at all for an automated test, it will cover driver's input for all permitted ages.

```
// return a number between 10 and 100  
driver.findElement(By.name("drivers_age")).sendKeys("'" + getRandomInteger(10, 100));
```

15.4 Get a random character

```
function getRandomCharacter() {  
  var chars = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";  
  return chars.substr( Math.floor(Math.random() * 62), 1);  
}
```

15.5 Get a random string at fixed length

```
function getRandomString(strLength) {  
  var text = "";  
  var possible = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";  
  
  for( var i=0; i < strLength; i++ )  
    text += possible.charAt(Math.floor(Math.random() * possible.length));  
  return text;  
}  
  
test.it("Random string for text field", function() {  
  driver.findElement(By.name("password")).sendKeys(getRandomString(8));  
});
```

By creating some utility functions (you can find in source project), we can get quite readable test scripts as below:

```
getRandomString(7); // example: "dolorem"  
getRandomwords(5); // example: "sit doloremque consequatur accusantium aut"  
getRandomSentences(3);  
getParagraphs(2);
```

15.6 Get a random string in a collection

```
function getRandomStringIn(array) {
  return array[Math.floor(Math.random() * array.length)];
}

test.it("Random string in collection", function() {
  var allowableStrings = ["Yes", "No", "Maybe"]; // one of these strings
  driver.findElement(By.name("username")).sendKeys(getRandomStringIn(allowableStrings));
});
```

I frequently use this in my test scripts.

15.7 Generate random person names, emails, addresses with Faker

[Faker³](#) is a Node.js library that generates fake data.

```
var faker = require('faker');
console.log(faker.internet.email()) // Johnathan_Lowe84@hotmail.com
console.log(faker.name.findName()) // Gus Stark
// generate based on formatting
console.log(faker.fake('{{name.firstName}} {{name.lastName}}'));
console.log(faker.address.streetAddress()); // 3205 Camylle Lights
console.log(faker.phone.phoneNumber()); // (709) 008-4599 x129

driver.findElement(By.name("username")).sendKeys(faker.internet.email());
```

You can find more examples at [Faker⁴](#) web site.

15.8 Generate a test file at fixed sizes

When testing file uploads, testers often try test files in different sizes. The following statements generates a test file in precise size on the fly.

³<https://www.npmjs.com/package/Faker>

⁴<https://github.com/FotoVerite/Faker.js>


```
var fs = require('fs');
var outputFile = __dirname + "/tmp/2MB.txt";
fs.writeFileSync(outputFile, "0".repeat(1024 * 1024 * 2));
```

15.9 Retrieve data from Database

The ultimate way to obtain accurate test data is to get from the database. For many projects, this might not be possible. For ones do, this provides the ultimate flexibility in terms of getting test data.

The test script example below is to enter the oldest (by age) user's login into the text field on a web page. To get this oldest user in the system, I use SQL to query a SQLite3 database directly (it will be different for yours, but the concept is the same) using [sqlite3 module](https://www.npmjs.com/package/sqlite3)⁵.

Install sqlite3 with npm first.

```
> npm install --save sqlite3
```

```
var fs = require('fs');
var sqlite3 = require('sqlite3').verbose();
var db_file = __dirname + "/../testdata/sample.db"
var db = new sqlite3.Database(db_file);

var oldestUserLogin = "";
db.serialize(function() {
  if (!fs.existsSync(db_file)) {
    console.log("DB not exists!")
  }

  db.get("SELECT login FROM users ORDER BY age DESC", function(err, row) {
    // console.log(row.login);
    oldestUserLogin = row.login;
    console.log("user => " + oldestUserLogin);
    driver.findElement(By.id("user")).sendKeys(oldestUserLogin);
  });
});

db.close();
```

⁵<https://www.npmjs.com/package/sqlite3>

16. Browser Profile and Capabilities

Selenium can start browser instances with various profile preferences which can be quite useful. Obviously, some preference settings are browser specific, so you might take some time to explore. In this chapter, I will cover some common usages.

16.1 Get browser type and version

Detecting browser type and version is useful to write custom test scripts for different browsers.

```
driver = new webdriver.Builder().forBrowser('chrome').build();
// console.log(driver.getCapabilities()); // returns ManagedPromise object
driver.getCapabilities().then(function(caps){
    assert.equal("chrome", caps.get("browserName")); // firefox/chrome/internet explorer
    console.log(caps.get("version")); // example: "46.0", "51.0.2704.103"
});
```

16.2 Set HTTP Proxy for Browser

Here is an example to set HTTP proxy server for Chrome.

```

var webdriver = require('selenium-webdriver'),
    proxy = require('selenium-webdriver/proxy');

var driver = new webdriver.Builder()
    .forBrowser('chrome')
    .setProxy(proxy.manual({http: 'proxy:3128'}))
    .build();

// if proxy not set up, shall not able to visit
driver.get("http://testwisely.com/demo").then(null, function(err) {
    console.log("Proxy error: " + err);
});

```

16.3 Verify file download in Chrome with Custom User Preferences

To efficiently verify a file is downloaded, we would like to

- save the file to a specific folder
- avoid “Open with or Save File” dialog

```

var chrome = require('selenium-webdriver/chrome');
var chromeOptions = new chrome.Options();
var path = require('path');
var tmp_dir = path.normalize(__dirname + "../tmp/");
chromeOptions.setUserPreferences({'download.default_directory': tmp_dir});
var driver = new webdriver.Builder()
    .forBrowser('chrome')
    .setChromeOptions(chromeOptions)
    .build();
driver.get("http://zhimin.com/books/pwta");
driver.findElement(By.linkText("Download")).click().then(function() {
    driver.sleep(10000); // wait 10 seconds for downloading to complete
    var fs = require('fs');
    var filePath = tmp_dir + "practical-web-test-automation-sample.pdf"
    assert(fs.statSync(filePath).isFile());
});

```

More Chrome preferences: http://src.chromium.org/svn/trunk/src/chrome/common/pref_names.cc¹

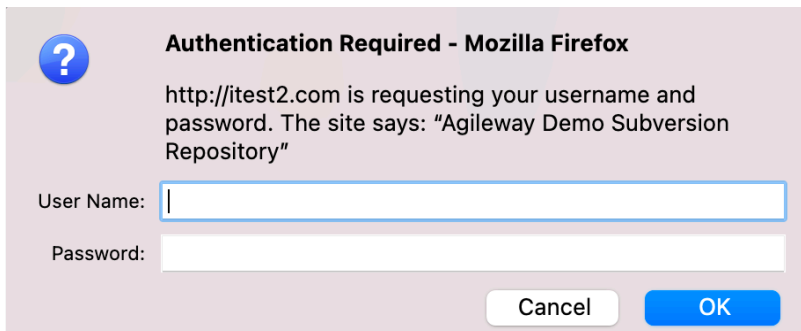
¹http://src.chromium.org/svn/trunk/src/chrome/common/pref_names.cc

16.4 Test downloading PDF in Firefox with Custom Profile

```
var firefox = require('selenium-webdriver/firefox')
var path = require('path');
var myDownloadFolder = path.normalize(__dirname + '/../tmp/');
var profile = new firefox.Profile();
profile.setPreference("browser.download.folderList", 2);
profile.setPreference("browser.download.dir", myDownloadFolder);
profile.setPreference("browser.helperApps.neverAsk.saveToDisk", "application/pdf");
// disable Firefox's built-in PDF viewer
profile.setPreference("pdfjs.disabled", true);
var options = new firefox.Options().setProfile(profile);
var driver = new webdriver.Builder().forBrowser('firefox')
    .setFirefoxOptions(options)
    .build();
driver.get("http://zhimin.com/books/pwta");
driver.findElement(By.linkText("Download")).click().then(function() {
    driver.sleep(10000); // wait 10 seconds for downloading to complete
    var fs = require('fs');
    var filePath = myDownloadFolder + "practical-web-test-automation-sample.pdf"
    assert(fs.statSync(filePath).isFile());
})
```

16.5 Bypass basic authentication by embedding username and password in URL

Authentication dialogs, like the one below, can be troublesome for automated testing.



The image shows a standard Firefox authentication dialog box. It has a light purple background and a blue question mark icon in the top left corner. The title bar reads "Authentication Required - Mozilla Firefox". The main text states: "http://itest2.com is requesting your username and password. The site says: 'Agileway Demo Subversion Repository'". Below this text are two input fields: "User Name:" and "Password:". At the bottom right, there are two buttons: "Cancel" and "OK".

A very simple way to get pass Basic or NTLM authentication dialogs: prefix username and password in the URL.

```
driver = new webdriver.Builder().forBrowser('firefox').build();
driver.get("http://tony:password@itest2.com/svn-demo/")
// got in, click a link
driver.findElement(By.linkText("tony/")).click();
```

16.6 Bypass basic authentication with Firefox AutoAuth plugin

There is another complex but quite useful approach to bypass basic authentication: use a browser extension. Take Firefox for example, “[Auto Login](#)”² submits HTTP authentication dialogs remembered passwords.

By default, Selenium starts Firefox with an empty profile, which means no remembered passwords and extensions. We can instruct Selenium to start Firefox with an existing profile.

- Start Firefox with a dedicated profile. Run the command below (from command line)

Windows:

```
"C:\Program Files (x86)\Mozilla Firefox\firefox.exe" -p
```

Mac:

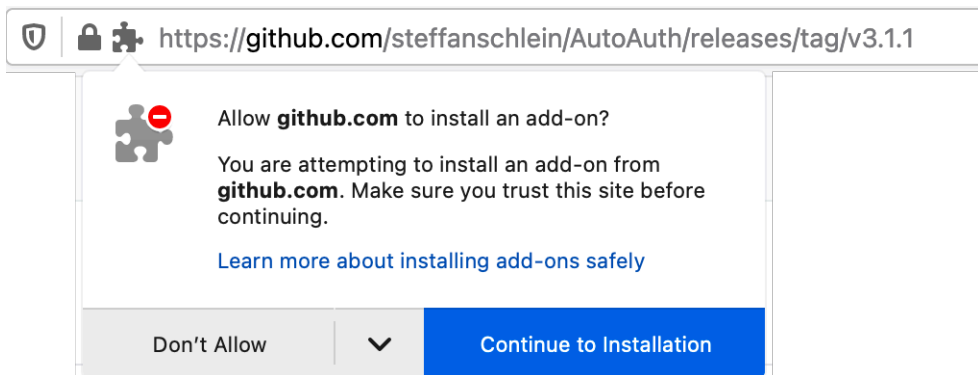
```
/Applications/Firefox.app/Contents/MacOS/firefox-bin -p
```

- Create a profile (I name it ‘testing’) and start Firefox with this profile

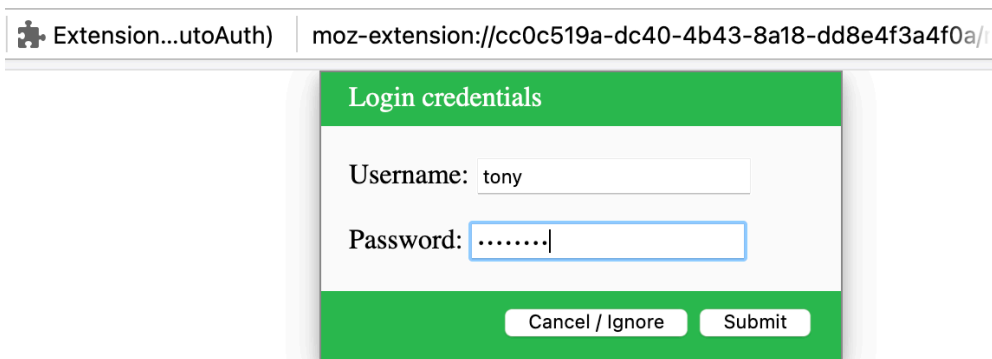
²<https://addons.mozilla.org/en-US/firefox/addon/autoauth/>



- Install autoauth plugin. A simple way: drag the file *autoauth-3.1.1-an.fx.xpi* (included with the test project) to Firefox window.



- Visit the web site requires authentication. Manually type the user name and password. Click 'Remember Password'.



Now the preparation work is done (and only need to be done once).

```
// Prerequisite: the password is already remembered in 'testing' profile.
var firefox = require('selenium-webdriver/firefox')
var AUTOAUTH_EXTENSION = __dirname + '/../autoauth-2.1-fx+fn.xpi'
var profile = new firefox.Profile();
// profile.setPreference('general.useragent.override', 'foo;bar');
profile.addExtension(AUTOAUTH_EXTENSION);
var options = new firefox.Options().setProfile(profile);
var driver = new webdriver.Builder().forBrowser('firefox')
    .setFirefoxOptions(options)
    .build();
driver.get("http://tony:password@itest2.com/svn-demo/")
driver.findElement(By.linkText("tony/")).click();
```

The 'testing' profile name must be supplied, otherwise it won't work.

16.7 Manage Cookies

```
driver.get("http://travel.agileway.net")
driver.manage().getCookies().then(function(cookies) {
    assert.equal(1, cookies.length);
});

driver.manage().addCookie({name: 'username', value: 'agileway'});
driver.manage().getCookies().then(function(cookies) {
    assert.equal(2, cookies.length);
});
```

16.8 Headless browser testing with PhantomJS

A headless browser is a web browser without a graphical user interface. The main benefit of headless browser testing is performance. [PhantomJS](http://phantomjs.org/)³ is a headless browser that built on top of WebKit, the engine behind both Safari and Chrome. First of all, you need to download phantomjs.exe and put it in PATH.

```
driver = new webdriver.Builder().forBrowser('phantomjs').build();
driver.get("http://travel.agileway.net")
driver.getTitle().then(function(the_title){
    assert.equal("Agile Travel", the_title)
});
driver.quit
```

If your target application is relatively stable and not using JavaScript heavily, and you want gain test faster execution time, PhantomJS is a viable option.

Be aware of headless browser simulation in test automation

Frankly, I am not big fan of headless testing with browser simulation for the reasons below:

- It is NOT a real browser.
- I need inspect the web page when a test failed, I cannot do that with PhantomJS. In test automation, as we know, we perform this all the time.
- To achieve faster execution time, I prefer distributing tests to multiple build agents to run them in parallel as a part of Continuous Testing process. That way, I get not only much faster execution time (throwing in more machines), also get useful features

³<http://phantomjs.org/>

such as quick feedback, rerunning failed tests on another build agent, dynamic execution ordering by priority, etc. All in real browsers.

Update [April 13, 2017]: [PhantomJS' maintainer stepped down](#)³, the future of PhantomJS is in doubt.

³<https://groups.google.com/forum/m/#!topic/phantomjs/9aI5d-LDuNE>

16.9 Headless Chrome

Chrome 59 (released on June 5, 2017) introduces headless mode, which can be used with Selenium WebDriver. Here is how:

```
var chrome = require('selenium-webdriver/chrome');
var chromeOptions = new chrome.Options();
chromeOptions.addArguments("headless");
var driver = new webdriver.Builder().forBrowser('chrome')
    .setChromeOptions(chromeOptions)
    .build();
driver.get("http://travel.agileway.net")
driver.getTitle().then(function(the_title){
    assert.equal("Agile Travel", the_title)
});
```

Please note that, at the time of writing, Headless mode is available on Mac and Linux in Chrome 59, but [Windows support](#)⁴ will be available soon.

16.10 Headless Firefox

Headless is also supported in Firefox from version 56, works the same way as Chrome. Please note the syntax has been changed several times, the latest one is `options.headless!`.

⁴<https://bugs.chromium.org/p/chromium/issues/detail?id=686608>

```
var firefox = require('selenium-webdriver/firefox');
var firefoxOptions = new firefox.Options();
firefoxOptions.headless();
var driver = new webdriver.Builder().forBrowser('firefox')
    .setFirefoxOptions(firefoxOptions)
    .build();
```

16.11 Test responsive web pages

Modern websites embrace responsive design to fit in different screen resolutions on various devices, such as iPad and smartphones. Bootstrap is a very popular responsive framework. How to verify your web site's responsiveness is a big question, it depends what you want to test. A quick answer is to use WebDriver's `driver.manage().window().setSize()` to set your browser to a target resolution, and then execute tests.

The example below verify a text box's width changes when switching from a desktop computer to a iPad, basically, whether responsive is enabled or not..

```
var driver = new webdriver.Builder().forBrowser('chrome').build();
driver.get("https://agileway.net")

// Desktop
var width_desktop;
driver.manage().window().setSize(1200, 800).then(function() {
    driver.findElement(By.name("email")).getSize().then(function(elemSize) {
        width_desktop = elemSize.width;
    });
});

// iPad
var width_ipad;
driver.manage().window().setSize(768, 1024).then(function() {
    driver.findElement(By.name("email")).getSize().then(function(elemSize) {
        width_ipad = elemSize.width;
        assert(width_desktop < width_ipad); // 1050 vs 358
    });
});
```

17. Advanced User Interactions

The Actions in Selenium WebDriver provides a way to set up and perform complex user interactions. Specifically, grouping a series of keyboard and mouse operations and sending to the browser.

Mouse interactions

The Actions mouse APIs in JavaScript are slightly different from Java's (and other language bindings).

Java	JavaScript
click()	click()
clickAndHold()	mouseDown().mouseMove(targetElem)..mouseUp()
contextClick()	click(elem, webdriver.Button.RIGHT)
doubleClick()	doubleClick()
dragAndDrop()	dragAndDrop()
dragAndDropBy()	dragAndDrop(location, {x: number, y: nubmer})
moveByOffset()	mouseMove(location, opt_offset)
moveToElement()	mouseMove()
release()	mouseUp()

Keyboard interactions

- keyDown()
- keyUp()
- sendKeys()

The usage

```
driver.actions(). + one or more above operations + .perform();
```

Check out the [WebDriver actions.js](https://github.com/SeleniumHQ/selenium/blob/9f51796fc96120b7aaebd91b90418e1a13fbab51/javascript/node/selenium-webdriver/lib/actions.js)¹ for more.

17.1 Double click a control

¹<https://github.com/SeleniumHQ/selenium/blob/9f51796fc96120b7aaebd91b90418e1a13fbab51/javascript/node/selenium-webdriver/lib/actions.js>

```
driver.get("file://" + __dirname + "/../../site/text_field.html");

var elem = driver.findElement(By.id("quickfill"));
driver.actions()
    .doubleClick(elem)
    .perform();
driver.findElement(By.id("pass")).getAttribute("value").then(function(pwd_value) {
    assert.equal("ABC", pwd_value);
}))
```

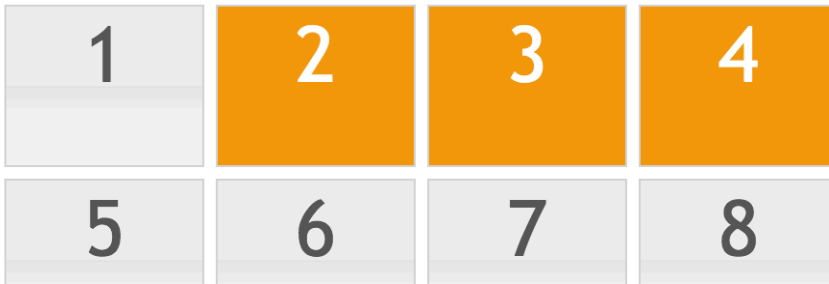
17.2 Move mouse to a control - Mouse Over

```
var elem = driver.findElement(By.id("email"));
driver.actions()
    .mouseMove(elem)
    .perform();
```

17.3 Click and hold - select multiple items

The test scripts below clicks and hold to select three controls in a grid.

```
driver.get("http://jqueryui.com/selectable");
driver.findElement(By.linkText("Display as grid")).click().then(function() {
    driver.sleep(2000);
    driver.switchTo().frame(0);
    driver.findElements(By.xpath("//ol[@id='selectable']/li")).then(function(lis){
        driver.actions()
            .mouseMove(lis[1])
            .mouseDown()
            .mouseMove(lis[3])
            .mouseUp()
            .perform();
        driver.switchTo().defaultContent();
    });
});
```

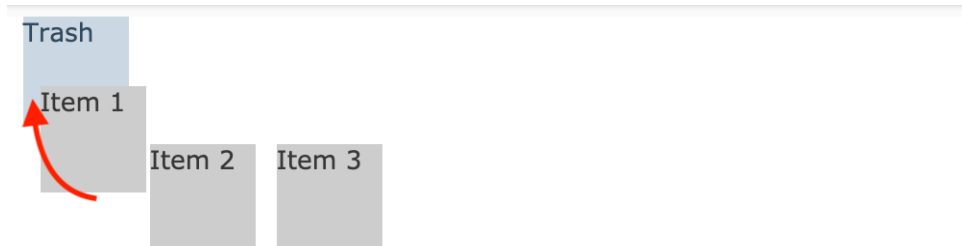


17.4 Context Click - right click a control

```
driver.get("file://" + __dirname + "/../../site/text_field.html");
driver.sleep(200);
var elem = driver.findElement(By.id("pass"));
driver.actions()
    .click(elem, webdriver.Button.RIGHT)
    .sendKeys(webdriver.Key.DOWN)
    .sendKeys(webdriver.Key.DOWN)
    .sendKeys(webdriver.Key.DOWN)
    .sendKeys(webdriver.Key.DOWN)
    .sendKeys(webdriver.Key.RETURN)
    .perform();
```

17.5 Drag and drop

Drag-n-drop is increasingly more common in new web sites. Testing this feature can be largely achieved in Selenium, I used the word ‘largely’ means achieving the same outcome, but not the ‘mouse dragging’ part. For this example page,



the test script below will *drop* ‘Item 1’ to ‘Trash’.

```
driver.get("file://" + __dirname + "../../../site/drag_n_drop.html");
var dragFrom = driver.findElement(By.id("item_1"));
var target = driver.findElement(By.id("trash"));

driver.actions()
    .mouseMove(dragFrom)
    .mouseDown()
    .mouseMove(target)
    .mouseUp()
    .perform();
```

The below is a screenshot after the test execution.



17.6 Drag slider

Slider (a part of JQuery UI library) provide users an very intuitive way to adjust values (typically in settings).

Slider



The test below simulates 'dragging the slider to the right'.

```

driver.findElement(By.id("pass_rate")).getText().then(function(the_text) {
    assert.equal("15%", the_text);
});

var elem = driver.findElement(By.id("pass-rate-slider"));
driver.actions()
    .dragAndDrop(elem, {x: 30, y: 0})
    .perform();

driver.findElement(By.id("pass_rate")).getText().then(function(the_text) {
    assert.notEqual("15%", the_text);
});

```

The below is a screenshot after the test execution.

Slider



Please note that the percentage figure after executing the test above are always 50% (I saw 49% now and then).

17.7 Send key sequences - Select All and Delete

```

driver.get("file://" + __dirname + "/../../site/text_field.html");
driver.findElement(By.id("comments")).sendKeys("Multiple Line\r\n Text");
var elem = driver.findElement(By.id("comments"));
driver.actions()
    .click(elem)
    .keyDown(webdriver.Key.CONTROL)
    .sendKeys("a")
    .keyUp(webdriver.Key.CONTROL)
    .perform();

// this different from click element, the key is send to browser directly
driver.actions()
    .sendKeys(webdriver.Key.BACK_SPACE)
    .perform();

```

Please note that the last test statement is different from `elem.sendKeys`. The keystrokes triggered by `Actions.sendKeys` is sent to the active browser window, not a specific element.

17.8 Click a specific part of an image

The image below contains special marked areas (appears on mouse over) linked to different URLs.



Here is the HTML:

```
<img src='images/software.png' id="agileway_software" usemap="#agileway_software_map">
<map name="agileway_software_map" id="agileway_software_map">
  <area shape="rect" coords="10,60,120,88" href="https://testwisely.com/buildwise"/>
  <area shape="circle" coords="200,25,18" href="https://clinicwise.net">
</map>
```

We can use `move_to(elem, xoffset, yoffset)` to click specific coordinates.

```
driver.get("file://" + __dirname + "/../../site/image-map.html");
elem = driver.findElement(By.id("agileway_software"));
driver.actions()
    .mouseMove(elem, {x: 190, y: 30} )
    .click()
    .perform();
driver.getTitle().then(function(the_title){
    assert.equal("ClinicWise - Cloud based Health Clinic Management System", the_title);
});
driver.get("file://" + __dirname + "/../../site/image-map.html");
elem = driver.findElement(By.id("agileway_software"));
driver.actions()
    .mouseMove(elem, {x: 30, y: 75} )
    .click()
    .perform();
driver.getTitle().then(function(the_title){
    assert.equal("BuildWise - TestWisely", the_title);
});
```


18. HTML 5 and Dynamic Web Sites


Web technologies are evolving. HTML5 includes many new features for more dynamic web applications and interfaces. Furthermore, wide use of JavaScript (thanks to popular JavaScript libraries such as JQuery), web sites nowadays are much more dynamic. In this chapter, I will show some Selenium examples to test HTML5 elements and interactive operations.

Please note that some tests only work on certain browsers (Chrome is your best bet), as some HTML5 features are not fully supported in some browsers yet.

18.1 HTML5 Email type field

Let's start with a simple one. An email type field is used for input fields that should contain an e-mail address. From the testing point of view, we treat it exactly the same as a normal text field.

Email field



HTML Source

```
<input id="email" name="email" type="email" style="height:30px; width: 280px;">
```

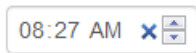
Test Script

```
driver.get("file://" + __dirname + "/../../site/html5.html");  
  
driver.findElement(By.id("email")).sendKeys("test@wisely.com");
```

18.2 HTML5 Time Field

The HTML5 time field is much more complex, as you can see from the screenshot below.

Time



HTML Source

```
<input id="start_time_1" name="start_time" type="time" style="width: 120px;">
```

The test scripts below do the following:

1. make sure the focus is not on this time field control
2. click and focus the time field
3. clear existing time
4. enter a new time

```
driver.findElement(By.id("start_time_1")).sendKeys("12:05AM");
```

```
// focus on another ...
```

```
driver.findElement(By.id("home_link")).sendKeys("");
driver.sleep(500);
```

```
// now back to change it
```

```
driver.findElement(By.id("start_time_1")).click();
// [:delete, :left, :delete, :left, :delete]
driver.findElement(By.id("start_time_1")).sendKeys(webdriver.Key.DELETE);
driver.findElement(By.id("start_time_1")).sendKeys(webdriver.Key.LEFT);
driver.findElement(By.id("start_time_1")).sendKeys(webdriver.Key.DELETE);
driver.findElement(By.id("start_time_1")).sendKeys(webdriver.Key.LEFT);
driver.findElement(By.id("start_time_1")).sendKeys(webdriver.Key.DELETE);
```

```
driver.findElement(By.id("start_time_1")).sendKeys("08");
driver.sleep(300);
driver.findElement(By.id("start_time_1")).sendKeys("27");
driver.sleep(300);
driver.findElement(By.id("start_time_1")).sendKeys("AM");
```

18.3 Invoke 'onclick' JavaScript event

In the example below, when user clicks on the text field control, the tip text ('Max 20 characters') is shown.

Example page

 Max 20 characters

HTML Source

```
<input type="text" name="person_name" onclick="$('#tip').show();"
onchange="change_person_name(this.value);"/>
<span id="tip" style="display:none; margin-left: 20px; color:gray;">
  Max 20 characters</span>
```

When we use normal `sendKeys()` in Selenium, it enters the text OK, but the tip text is not displayed.

```
driver.findElement(By.name("person_name")).sendKeys("Wise Tester");
```

We can simply calling 'click' to achieve it.

```
driver.findElement(By.name("person_name")).clear();
driver.findElement(By.name("person_name")).sendKeys("Wise Tester");
driver.findElement(By.name("person_name")).click();
driver.findElement(By.id("tip")).getText().then(function(tipText) {
    assert.equal("Max 20 characters", tipText);
});
```

18.4 Invoke JavaScript events such as 'onchange'

A generic way to invoke 'OnXXXX' events is to execute JavaScript, the below is an example to invoke 'OnChange' event on a text box.

```
driver.findElement(By.name("person_name")).sendKeys("Wise Tester too");
driver.executeScript("$('#person_name_textbox').trigger('change)').then(function() {
    driver.findElement(By.id("person_name_label")).getText().then(function(theText){
        assert.equal("Wise Tester too", theText);
    });
});
```

18.5 Scroll to the bottom of a page

Calling JavaScript API.

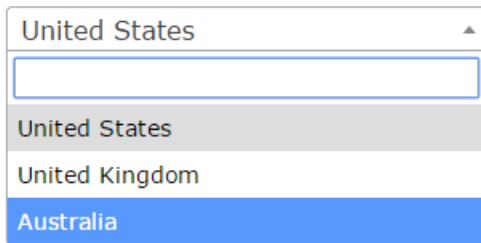
```
driver.executeScript("window.scrollTo(0, document.body.scrollHeight);").then(function(){  
});
```

Or send the keyboard command: 'Ctrl+End'.

```
driver.findElement(By.tagName("body")).sendKeys(webdriver.Key.CONTROL, webdriver.Key.END);
```

18.6 Select2 - Single Select

Select2 is a popular JQuery plug-in that makes long select lists more user-friendly, it turns the standard HTML select list box into this:



HTML Source

```
<select id="country_single" data-placeholder="Choose a Country..." class="select2">  
  <option value=""></option>  
  <option value="United States">United States</option>  
  <option value="United Kingdom">United Kingdom</option>  
  <option value="Australia">Australia</option>  
</select>
```

The HTML source seems not much different from the standard select list excepting adding the class `select2`. By using the class as the identification, the JavaScript included on the page generates the following HTML fragment (beneath the select element).

Generated HTML Source

```

<select id="country_single" .... />
<span class="select2-container select2-container--default select2-container--open"><span \
class="select2-dropdown select2-dropdown--below"><span class="select2-search select2-sear\
ch--dropdown">
    <input autocomplete="off" class="select2-search__field" type="search">
</span><span class="select2-results"></span></span></span>
<ul class="select2-results__options" id="select2-country_single-results">
    <li class="..." id="..."><span class="select2-results">United States</span></li>
    <li class="..." id="..."><span class="select2-results">United Kingdom</span></li>
    <li class="..." id="..."><span class="select2-results">Australia</span></li>
</ul>

```

Please note that this dynamically generated HTML fragment is not viewable by ‘View Page Source’, you need to enable the inspection tool (usually right mouse click the page, then choose ‘Inspect Element’) to see it.

Before we test it, we need to understand how we use it.

- Click the ‘Choose a Country’
- Select an option

There is no difference from the standard select list. That’s correct, we need to understand how Select2 emulates the standard select list first. In Select2, clicking the ‘Choose a Country’ (to show the options) is actually clicking the span next to the original <select> with id country_single; selecting an option is clicking a list item (tag: li) under ul with id select2-country_single-results (where country_single is the ID of original Select). With this knowledge, plus XPath in Selenium, we can drive a Select2 standard select box with the test scripts below:

```

driver.get("file://" + __dirname + "../..../site/select2.html");
driver.sleep(500); // wait JS to load

driver.findElement(By.xpath("//select[@id='country_single']/../span")).click();
driver.sleep(300);
driver.findElements(By.xpath("//ul[@id='select2-country_single-results']/li")).then(
    function(options){
        options.forEach(function(option){
            option.getText().then(function(optionText) {
                if (optionText == "Australia") {
                    option.click();
                }
            });
        });
    });

```

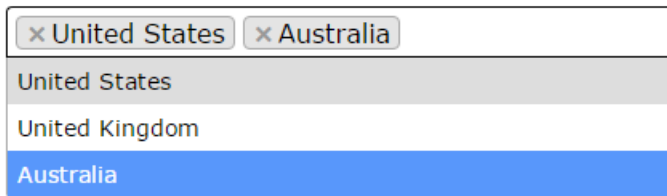
```
});
});
```

A neat feature of Select2 is allowing user to search the option list, to do that in Selenium:

```
driver.findElement(By.xpath("//select[@id='country_single']/../span//span[@class='select2\
-selection__arrow']")).click();
driver.sleep(300);
var search_text_field = driver.findElement(By.xpath("//span/input[@class = 'select2-search\
h__field']"));
search_text_field.sendKeys("United King")
driver.sleep(200); // let filtering finishing
search_text_field.sendKeys(webdriver.Key.ENTER) // select 1st highlighted
```

18.7 Select2 - Multiple Select

Select2¹ also enhances the multiple selection (a lot).



HTML Source

```
<select id="country_multiple" data-placeholder="Choose Countries..." class="select2"
      multiple style="width:420px;">
  <option value=""></option>
  <option value="United States">United States</option>
  <option value="United Kingdom">United Kingdom</option>
  <option value="Australia">Australia</option>
</select>
```

Again, the only difference from the standard multiple select list is the class 'select2'. Astute readers will find the generated HTML fragment is different from the standard (single) select, that's because of the usage. The concept of working out driving the control is the same, I will leave the homework to you, just show the test scripts.

¹<https://github.com/select2/select2>

```

var containerXpath = "//select[@id='country_multiple']/../span[contains(@class, 'select2-\
container')]";
driver.findElement(By.xpath(containerXpath)).click();
driver.sleep(200);

driver.findElements(By.xpath("//ul[@id='select2-country_multiple-results']/li")).then(
    function(options){
        options.forEach(function(option){
            option.getText().then(
                function(optionText) {
                    if (optionText == "United States") {
                        option.click();
                        return;
                    }
                },
                function(err) {
                    // need to handle, after click, the options is stale
                });
            });
    });

```

To deselect an option is to click the little 'x' on the right. In fact, it is the idea to clear all selections first then select the wanted options.

```

driver.findElements(By.xpath(removeBtnXpath)).then(function(closeBtns) {
    var flagCleared = false
    for (i = 0; i < closeBtns.length; i++) {
        driver.findElement(By.xpath(removeBtnXpath)).click();
        flagCleared = true
    }
    if (flagCleared) {
        driver.findElement(By.xpath(containerXpath)).click()
    }
});

```

Some might say the test scripts are quite complex. That's good thinking, if many of our test steps are written like this, it will be quite hard to maintain. One common way is to extract them into reusable functions, like below:

```

function select2_multiple_clear(elem_id) {
    driver.sleep(500); // wait JS to load
    var containerXpath = "//select[@id='" + elem_id + "']/../span[contains(@class, 'select2\
-container')]";
    var removeBtnXpath = containerXpath + "//span[@class='select2-selection__choice__remove\
']";
    driver.findElements(By.xpath(removeBtnXpath)).then(function(closeBtns) {
        var flagCleared = false
        for (i = 0; i < closeBtns.length; i++) {
            driver.findElement(By.xpath(removeBtnXpath)).click();
            flagCleared = true
        }
        if (flagCleared) {
            driver.findElement(By.xpath(containerXpath)).click()
        }
    });
    driver.sleep(200)
}

function select2_multiple(elem_id, oneOptionText) {
    driver.sleep(500); // wait JS to load
    var containerXpath = "//select[@id='" + elem_id + "']/../span[contains(@class, 'select2\
-container')]";
    driver.findElement(By.xpath(containerXpath)).click();
    driver.sleep(300);
    driver.findElements(By.xpath("//ul[@id='select2-" + elem_id + "-results']/li")).then(
        function(options){
            options.forEach(function(option){
                option.getText().then(function(optionText) {
                    console.log("> " + optionText);
                    if (oneOptionText == optionText) {
                        option.click();
                        return;
                    }
                },
            ),
            function(err) {
                // after click, the options is stale
            });
        });
    driver.sleep(100);
}

```



```
// ...

test.it("Select2 calling functions", function() {
  select2_multiple("country_multiple", "United States");
  select2_multiple("country_multiple", "United Kingdom");
  select2_multiple_clear("country_multiple")
  select2_multiple("country_multiple", "United States");
});
```

You can find more techniques for writing maintainable tests from my other book *Practical Web Test Automation*².

18.8 AngularJS web pages

AngularJS is a popular client-side JavaScript framework that can be used to extend HTML. Here is a web page (simple TODO list) developed in AngularJS.

1 of 2 remaining [[archive](#)]

- ☒ learn angular
- ☐ build an angular app

HTML Source

The page source (via “View Page Source” in browser) is different from what you saw on the page. It contains some kind of dynamic coding (*ng-xxx*).

```
<div ng-controller="TodoCtrl">
  <span>{{remaining()}} of {{todos.length}} remaining</span>
  [ <a href="" ng-click="archive()">archive</a> ]
  <ul class="unstyled">
    <li ng-repeat="todo in todos">
      <input type="checkbox" ng-model="todo.done">
      <span class="done-{{todo.done}}">{{todo.text}}</span>
    </li>
  </ul>
  <form ng-submit="addTodo()">
    <input type="text" ng-model="todoText" size="30"
```

²<https://leanpub.com/practical-web-test-automation>

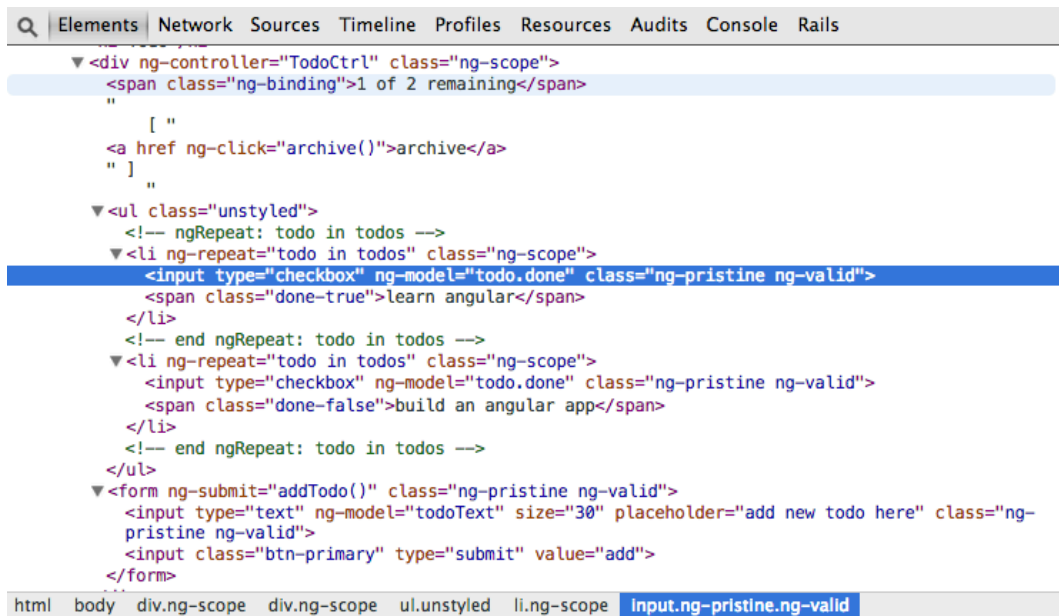
```

        placeholder="add new todo here">
    <input class="btn-primary" type="submit" value="add">
  </form>
</div>

```

As a tester, we don't need to worry about AngularJS programming logic in the page source. To view rendered page source, which matters for testing, inspect the page via right mouse click page and select "Inspect Element".

Browser inspect view



Astute readers will notice that the 'name' attribute are missing in the input elements, replaced with 'ng-model' instead. We can use xpath to identify the web element.

The tests script below

- Add a new todo item in a text field
- Click add button
- Uncheck the 3rd todo item

```

driver.getPageSource().then(function(the_page_source) {
    assert(the_page_source.contains("1 of 2 remaining"));
});

driver.findElement(By.xpath("//input[@ng-model='todoText']")).sendKeys("Learn Appium");
driver.findElement(By.xpath("//input[@type = 'submit' and @value='add']")).click();
driver.sleep(300);
driver.findElements(By.xpath("//input[@type = 'checkbox' and @ng-model='todo.done']").
then(function(elems){
    elems[2].click();
});
driver.sleep(500);

driver.getPageSource().then(function(the_page_source) {
    assert(the_page_source.contains("1 of 3 remaining"));
});

```

18.9 Ember JS web pages

Ember JS is another JavaScript web framework, like Angular JS, the ‘Page Source’ view (from browser) of a web page is raw source code, which is not useful for testing.

HTML Source

```

<div class="control-group">
  <label class="control-label" for="longitude">Longitude</label>
  <div class="controls">
    {{view Ember.TextField valueBinding="longitude"}}
  </div>
</div>

```

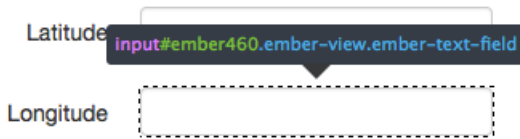
Browser inspect view

```

<label class="control-label" for="longitude">
  Longitude
</label>
<div class="controls">
  <input id="ember412" class="ember-view ember-text-field" type="text"></input>
</div>

```

The ID attribute of a Ember JS generated element (by default) changes. For example, this text field ID is “ember412”.



Refresh the page, the ID changed to a different value.



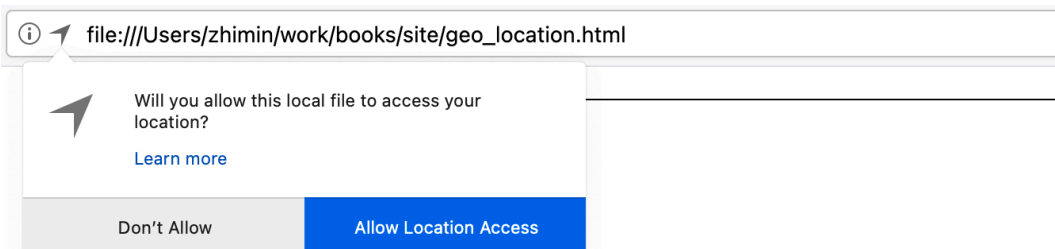
So we shall use another way to identify the element.

```
driver.get("file://" + __dirname + "/../../site/emberjs-crud-rest/index.html");
driver.findElement(By.linkText("Locations")).click();
driver.findElement(By.linkText("New location")).click();
driver.findElements(By.xpath("//div[@class='controls']/input[@class='ember-view ember-text-field']")).then(function(ember_text_fields) {
    ember_text_fields[0].sendKeys("-24.0034583945");
    ember_text_fields[1].sendKeys("146.903459345");
    ember_text_fields[2].sendKeys("90%");
});

driver.findElement(By.xpath("//button[text()='Update record']")).click();
```

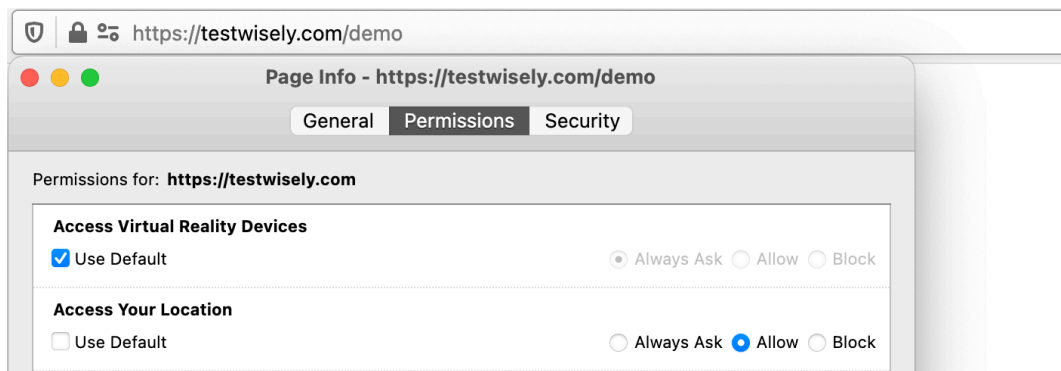
18.10 “Share Location” with Firefox

HTML5 Geolocation API can obtain a user’s position. By using Geolocation API, programmers can develop web applications to provide location-aware services, such as locating the nearest restaurants. When a web page wants to use a user’s location information, the user is presented with a pop up for permission.



This is a native popup window, which means Selenium WebDriver cannot drive it. However, there is a workaround. We can set up a browser profile that pre-allows “Share Location” for specific websites. Here are the steps for Firefox.

1. Open Firefox with a specific profile for testing
2. Open the site
3. Right click on page, select ‘View Page Info’ and select ‘Permissions’ (for versions before Firefox 45, type about:permissions in the address)
4. Select the site and choose “Allow” option for “Share Location”



The set up and use of a specific testing profile for Firefox is already covered in Chapter 16. This only needs to be done once. After that, the test script can test location-aware web pages.

```
var firefox = require('selenium-webdriver/firefox')
// dir for 'testing' profile, the uniq 8 characters are randomly created
var profileDir = "C:/Users/you/AppData/Roaming/Mozilla/Firefox/Profiles/xuzz1bwf.testing"
var profile = new firefox.Profile(profileDir);
var opts = new firefox.Options().setProfile(profile);
var driver = new webdriver.Builder().forBrowser('firefox')
    .setFirefoxOptions(opts)
    .build();
driver.get("http://testwisely.com/demo/geo-location");

driver.findElement(By.id("use_current_location_btn")).click();
driver.sleep(3000);
driver.findElement(By.id("demo")).getText().then(function(the_text){
    assert(the_text.contains("Latitude:"));
});
```

18.11 Faking Geolocation with JavaScript

With Geolocation testing, it is almost certain that we will need to test the users in different locations. This can be done by JavaScript.

```
var lati = "-34.915379";
var longti = "138.576777";
var js = "window.navigator.geolocation.getCurrentPosition=function(success){; var position = {'coords' : {'latitude': '"' + lati + "','longitude': '"' + longti + '"}}; success(position)};";
driver.executeScript(js).then(function(){
  driver.findElement(By.id("use_current_location_btn")).click();
  driver.sleep(1000);
  driver.findElement(By.id("demo")).getText().then(function(the_text){
    assert(the_text.contains("-34.915379"));
  });
});
```

18.12 Save a canvas to PNG image

HTML Canvas is used to draw graphics, typically via JavaScript. We can save the graphic in a canvas element to a PNG image file for test verification purposes.

```
driver.get("https://www.chartjs.org/docs/latest/samples/bar/vertical.html");
driver.sleep(1000); // wait canvas is loaded
canvas_elem = driver.findElement(By.xpath("//div/canvas"));
// the javascript to get image data
let js_str = "return arguments[0].toDataURL('image/png').substring(21);";
driver.executeScript(js_str, canvas_elem).then(function(canvas_base64) {
  var path = require('path');
  var filePath = path.normalize(__dirname + '/../tmp/saved_canvas.png');
  var fs = require('fs');
  fs.writeFile(filePath, canvas_base64, 'base64', function(err){
    if (err) throw err
    assert(fs.existsSync(filePath));
    console.log('File saved.')
  });
});
```

18.13 Verify dynamic charts

Dynamic charts on a web page are commonly drawn in a HTML canvas. To verify a chart is updated regularly, we can extract several chart image data at different time and compare them.

```
driver.get("file://" + __dirname + "../..../site/canvas.html");
canvas_elem = driver.findElement(By.id("myChart"))
driver.sleep(1000); // wait JS to load into Canvas
assert(canvas_elem.isDisplayed());

js_str = "return arguments[0].toDataURL('image/png').substring(21);";
var canvas_image_size_1 = canvas_image_size_2 = canvas_image_size_3 = 0;
driver.executeScript(js_str, canvas_elem).then(function(canvas_base64) {
    canvas_image_size_1 = canvas_base64.length;
});

driver.sleep(2000);
driver.executeScript(js_str, canvas_elem).then(function(canvas_base64) {
    canvas_image_size_2 = canvas_base64.length;
});

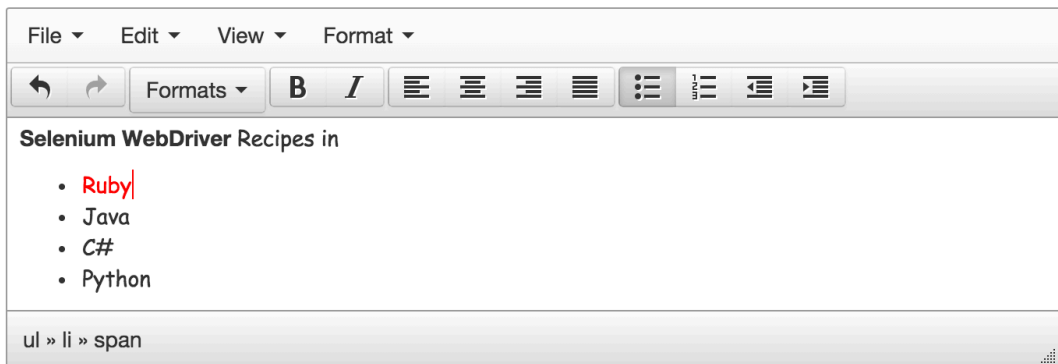
driver.sleep(2000);
driver.executeScript(js_str, canvas_elem).then(function(canvas_base64) {
    canvas_image_size_3 = canvas_base64.length;
    // verify the image of canvas all different (in size)
    assert(canvas_image_size_1 != canvas_image_size_2 != canvas_image_size_3);
});
```

19. WYSIWYG HTML editors

WYSIWYG (an acronym for “What You See Is What You Get”) HTML editors are widely used in web applications as embedded text editor nowadays. In this chapter, we will use Selenium WebDriver to test several popular WYSIWYG HTML editors.

19.1 TinyMCE

TinyMCE is a web-based WYSIWYG editor, it claims “the most used WYSIWYG editor in the world, it is used by millions”¹.



The rich text is rendered inside an inline frame within TinyMCE. To test it, we need to “switch to” that frame.

¹<http://www.tinymce.com/enterprise/using.php>


```
driver.get("file://" + __dirname + "../../../site/tinymce-4.1.9/tinyice_demo.html");

var tinymceFrame = driver.findElement(By.id("mce_0_ifr"));
driver.switchTo().frame(tinymceFrame);
var editorBody = driver.findElement(By.css("body"));
driver.executeScript("arguments[0].innerHTML = '<h1>Heading</h1>AgileWay'", editorBody);
driver.sleep(1000);
editorBody.sendKeys("New content");
driver.sleep(1000);
editorBody.clear();

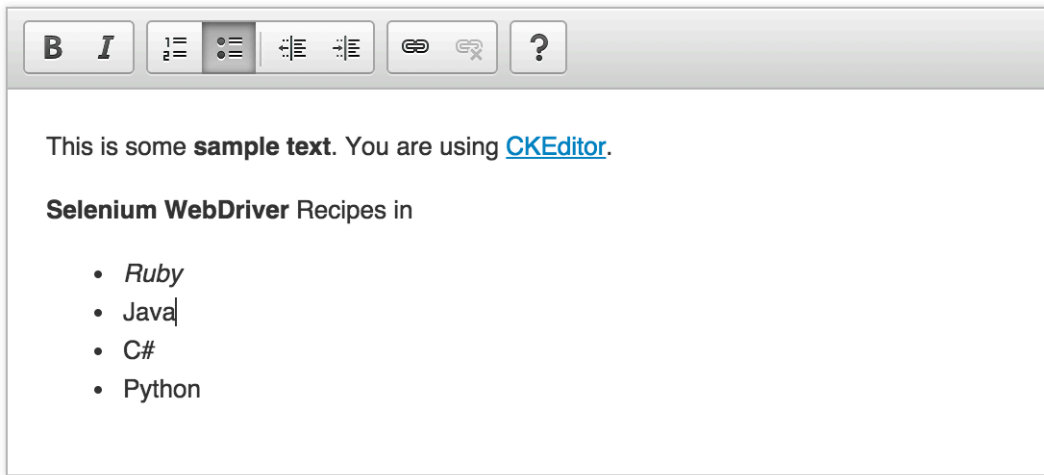
// click TinyMCE editor's 'Numbered List' button
driver.executeScript("arguments[0].innerHTML = '<p>one</p><p>two</p>'", editorBody);

// switch out then can drive controls on the main page
driver.switchTo().defaultContent();
var tinymceNumberListBtn = driver.findElement(By.css(".mce-btn[aria-label='Numbered list'\n] button"));
tinymceNumberListBtn.click();

// Insert
driver.executeScript("tinyMCE.activeEditor.insertContent('<p>Brisbane</p>')");
```

19.2 CKEditor

CKEditor is another popular WYSIWYG editor. Like TinyMCE, CKEditor uses an inline frame.



```
driver.get("file://" + __dirname + "/../../site/ckeditor-4.4.7/samples/uicolor.html");
driver.sleep(1000);

var ckeditorFrame = driver.findElement(By.className("cke_wysiwyg_frame"));
driver.switchTo().frame(ckeditorFrame);
var editorBody = driver.findElement(By.tagName("body"));
editorBody.sendKeys("Selenium Recipes\n by Zhimin Zhan");
driver.sleep(1000);

// Clear content another Method: using advanced user actions
driver.actions()
    .click(editorBody)
    .keyDown(webdriver.Key.CONTROL)
    .sendKeys("a")
    .keyUp(webdriver.Key.CONTROL)
    .perform();

driver.sleep(400)
driver.actions()
    .sendKeys(webdriver.Key.BACK_SPACE)
    .perform();

// switch out then can drive controls on the main page
driver.switchTo().defaultContent();
driver.findElement(By.className("cke_button_numberedlist")).click(); // numbered list
```

19.3 SummerNote

SummerNote is a Bootstrap based lightweight WYSIWYG editor, different from TinyMCE or CKEditor, it does not use frames.



Selenium WebDriver Recipes in

- Ruby
- Java
- C#
- Python

```
driver.get("file://" + __dirname + "/../../site/summernote-0.6.3/demo.html");
driver.sleep(500);
driver.findElement(By.xpath("//div[@class='note-editor']/div[@class='note-editable']")).sendKeys("Text");
// click a format button: unordered list
driver.findElement(By.xpath("//button[@data-event='insertUnorderedList']")).click();
// switch to code view
driver.findElement(By.xpath("//button[@data-event='codeview']")).click();
// insert code (unformatted)
driver.findElement(By.xpath("//textarea[@class='note-codable']")).sendKeys("\n<p>HTML</p>");
```

19.4 CodeMirror

CodeMirror is a versatile text editor implemented in JavaScript. CodeMirror is not a WYSIWYG editor, but it is often used with one for editing raw HTML source for the rich text content.

```
1 <!-- write some xml below -->
2 <Selenium-WebDriverRecipes>
3   <book>in Ruby</book>
4   <book>in Java</book>
5   <book>in C#</book>
6   <book>in Python</book>
7 </
  </Selenium-WebDriverRecipes>
```

```
driver.get("file://" + __dirname + "../../../site/codemirror-5.1/demo/xmlcomplete.html");
driver.sleep(200);
var elem = driver.findElement(By.className("CodeMirror-scroll"));
elem.click();
driver.sleep(500);
// elem.sendKeys does not work
driver.actions()
    .sendKeys("<h3>Heading 3</h3><p>TestWise is Selenium IDE</p>")
    .perform();
```

20. Leverage Programming

The reason that Selenium WebDriver quickly overtakes other commercial testing tools (typically promoting record-n-playback), in my opinion, is embracing the programming, which offers the flexibility needed for maintainable automated test scripts.

In the chapter, I will show some examples that use some programming practices to help our testing needs.

20.1 Raise exceptions to fail test

While `assert` provides most of assertions needed, raising exceptions can be useful too as shown below.

```
if (process.platform != "darwin" && process.platform != "linux") {  
    throw("Unsupported platform: " + process.platform);  
}
```

In test output (when running on Windows):

```
Error: done() invoked with non-Error: Unsupported platform: win32
```

An exception means an anomalous or exceptional condition occurred. The code to handle exceptions is called exception handling, an important concept in programming. If an exception is not handled, the program execution will terminate with the exception displayed.

Here is a standard way of handing exceptions in JavaScript.

```
try {
  driver.actions.sendKeys("TestWise").perform();
} catch (err) {
  console.log("Error occurred: " + err);
} finally {
  console.log("Clean up");
  driver.quit();
}
```

The above script exits successfully with the output:

```
Error occurred: TypeError: driver.actions.sendKeys is not a function
Clean up
```

catch block handles the exception. If an exception is handled, the program (in our case, test execution) continues. `ex.printStackTrace()` prints out the stack trace of the exception occurred. finally block is always run (after) no matter exceptions are thrown (from try) or not.

However, in Selenium WebDriver, we usually handle exceptions due to the use of JavaScript Promises.

```
driver.findElement(By.name("user")).sendKeys("bomb").then(null, function(err) {
  if (err.name == "NoSuchElementException") {
    console.log("Error: " + err);
  }
});
```

I often use exceptions in my test scripts for non-assertion purposes too.

1. Flag incomplete tests

The problem with “TODO” comments is that you might forget them.

```
test.it("Foo", function() {
  // TODO
});
```

I like this way better.

```
test.it("Foo", function() {  
  throw("TO BE DONE");  
});
```

2. Stop test execution during debugging a test

Sometimes, you want to utilize automated tests to get you to a certain page in the application quickly.

```
// test steps ...  
throw("Stop here, I take over from now. I delete this later.")
```

20.2 Ignorable test statement error

When a test step can not be performed correctly, execution terminates and the test is marked as failed. However, failed to run certain test steps sometimes is OK. For example, we want to make sure a test starts with no active user session. If a user is currently signed in, try signing out; If a user has already signed out, performing signing out will fail. But it is acceptable.

Here is an example to capture the error/failure in a test statement, and then ignore:

```
driver.findElement(By.name("mayExists")).click().then(null, function(err) {  
  // ignore the error if there is  
});  
// ...
```

20.3 Read external file

We can use Node.js's built-in file i/o functions to read data, typically test data, from external files. Try to avoid referencing an external file using absolute path like below:

```
fs = require('fs');  
// Read file in synchronously (blocking)  
var contents = fs.readFileSync("c:\\agileway\\in.xml", 'utf8'); // bad  
console.log(contents);  
// ...
```

If the test scripts is copied to another machine, it might fail. When you have a lot references to absolute file paths, it is going to be difficult to maintain. A common practice is to retrieve the test data folder from a reusable function (so that need to only update once).

```
fs = require('fs');
var input_file = __dirname + "../testdata/in.xml";
var contents = fs.readFileSync(input_file, 'utf8');
```

20.4 Data-Driven Tests with CSV

A CSV (comma-separated values) file stores tabular data in plain-text form. CSV files are commonly used for importing into or exporting from applications. Comparing to Excel spreadsheets, a CSV file is a text file that contains only the pure data, not formatting.

The below is the CSV version of data driving test for the above user sign in example, using [fast-csv¹](https://www.npmjs.com/package/fast-csv) library:

```
// npm install --save fast-csv
var csv = require("fast-csv");
fs = require('fs');
var input_file = __dirname + "../testdata/users.csv"
csv.parseFile(input_file).on("data", function(data){
  if (data[0] != "DESCRIPTION" ) { // ignore header row
    var login = data[1];
    var password = data[2];
    var expected_text = data[3];
    driver.get("http://travel.agileway.net")
    driver.findElement(By.name("username")).sendKeys(login);
    driver.findElement(By.name("password")).sendKeys(password);
    driver.findElement(By.name("commit")).click();
    driver.findElement(By.tagName("body")).getText().then(function(pageText) {
      assert(pageText.contains(expected_text));
    })

    driver.findElement(By.linkText("Sign off")).click().then(null, function(err) {
      // ignore the error if there is
    });
  }
})
.on("end", function(){
  console.log("done");
});
```

¹<https://www.npmjs.com/package/fast-csv>

20.5 Identify element IDs with dynamically generated long prefixes

You can use regular expression to identify the static part of element ID or NAME. The below is a HTML fragment for a text box, we could tell some part of ID or NAME are machine generated (which might be different for next build), and the part “AppName” is meaningful.

```
<input id="ctl00_m_g_dcb0d043_e7f0_4128_99c6_71c113f45dd8_ctl00_tAppName_I"
name="ctl00$m$g_dcb0d043_e7f0_4128_99c6_71c113f45dd8$ctl00$tAppName"/>
```

If we can later verify that ‘AppName’ is static for each text box, the test scripts below will work. Basically it instructs Selenium to find element whose name attribute contains “tAppName” (Watir can use Regular expression directly in finder, which I think it is better).

```
driver.get("file://" + __dirname + "../../../site/text_field.html");
var elemByName = driver.findElement(By.name("ctl00$m$g_dcb0d043_e7f0_4128_99c6_71c113f45d\nd8$ctl00$tAppName"));
elemByName.sendKeys("full name");
elemByName.clear();
driver.sleep(1000);
driver.findElement(By.xpath("//input[contains(@name, 'tAppName')]")).sendKeys("Still OK");
```

20.6 Sending special keys such as Enter to an element or browser

You can use .sendKeys() method to send special keys (and combination) to an web control.

```
driver.get("file://" + __dirname + "../../../site/text_field.html");
var elem = driver.findElement(By.id("user"));
elem.clear();
elem.sendKeys("agileway");
driver.sleep(1); // sleep for seeing the effect

elem.sendKeys(webdriver.Key.CONTROL, "a");
elem.sendKeys(webdriver.Key.BACK_SPACE);
driver.sleep(1000);
elem.sendKeys("testwisely");
driver.sleep(1000);
elem.sendKeys(webdriver.Key.ENTER); // submit the form
```

Some common special keys:

BACK_SPACE
DELETE
TAB
CONTROL
SHIFT
ALT
PAGE_UP
ARROW_DOWN
HOME
END
ESCAPE
ENTER
META
COMMAND

The full list can be found at Selenium Github: [selenium/javascript/webdriver/key.js](https://github.com/SeleniumHQ/selenium/blob/c10e8a955883f004452cdde18096d70738397788/javascript/webdriver/key.js)².

20.7 Use of Unicode in test scripts

Selenium WebDriver does support Unicode.

```
driver.get("file://" + __dirname + "/../../site/assert.html")

// assert unicode
driver.findElement(By.id("unicode_test")).getText().then(function(elemText) {
  assert.equal("𐄂", elemText);
});

// type unicode
driver.findElement(By.id("user")).sendKeys("𐄂𐄂𐄂𐄂𐄂𐄂𐄂");
```

20.8 Extract a group of dynamic data : verify search results in order

The below is a sortable table, i.e., users can sort table columns in ascending or descending order by clicking the header.

²<https://github.com/SeleniumHQ/selenium/blob/c10e8a955883f004452cdde18096d70738397788/javascript/webdriver/key.js>

Product	Released	URL
BuildWise	2010	https://testwisely.com/buildwise
ClinicWise	2013	https://clinicwise.net
SiteWise CMS	2014	http://sitewisecms.com
TestWise	2007	https://testwisely.com/testwise

To verify sorting, we need to extract all the data in the sorted column then verify the data in desired order. Knowledge of coding with List or Array is required.

```
// npm install --save async
var async = require('async');

driver.get("file://" + __dirname + "/../../site/data_grid.html");
driver.findElement(By.id("heading_product")).click(); // sort asc

var productNames = [];
var addCellText = function (elem, doneCallback) {
  elem.getText().then(function (textValue) {
    productNames.push(textValue);
    return doneCallback(null);
  });
};

driver.findElements(By.xpath("//tbody/tr/td[1]")).then(function (elems) {
  async.each(elems, addCellText, function (err) {
    // console.log("All Finished!");
    // console.log(productNames);
    var sortedProductNames = Array.prototype.slice.call(productNames).sort();
    assert.deepEqual(sortedProductNames, productNames);
  });
});
```

This approach is not limited to data in tables. The below script extracts the scores from the elements like `98`.

```
scoreElems = driver.findElements(By.xpath("//div[@id='results']/span[@class='score']"));
// ...
```

20.9 Verify uniqueness of a set of data

Like the recipe above, extract data and store them in an array first, then compare the number of elements in the array with another one without duplicates.

```
driver.findElements(By.xpath("//tbody/tr/td[2]")).then(function(elems){
  async.each(elems, addCellText, function(err) {
    var uniqYears = Array.from(new Set(releaseYears));
    assert.equal(uniqYears.length, releaseYears.length);
  });
});
```

20.10 Extract dynamic visible data rows from a results table

Many web search forms have filtering options that hide unwanted result entries.

☐ Test automation products only

Product	Released	URL	
ClinicWise	2013	https://clinicwise.net	<button>Like</button>
BuildWise	2010	https://testwisely.com/buildwise	<button>Like</button>
SiteWise CMS	2014	http://sitewisecms.com	<button>Like</button>
TestWise	2007	https://testwisely.com/testwise	<button>Like</button>
Displaying 1 - 4 of 4			

The test scripts below verify the first product name and click the corresponding 'Like' button.

```

driver.get("file://" + __dirname + "../../../site/data_grid.html");

driver.findElements(By.xpath("//table[@id='grid']/tbody/tr")).then(function(rows) {
    assert.equal(4, rows.length)
});

driver.findElement(By.xpath("//table[@id='grid']/tbody/tr[1]/td[1]")).getText().then(
    function(productName) {
        assert.equal("ClinicWise", productName)
    })
driver.findElement(By.xpath("//table[@id='grid']/tbody/tr[1]/td/button")).click();

```

Now check “Test automation products only” checkbox, and only two products are shown.

☒ Test automation products only

Product	Released	URL	
BuildWise	2010	https://testwisely.com/buildwise	Like
TestWise	2007	https://testwisely.com/testwise	Like
Displaying 1 - 4 of 4			

```

driver.findElement(By.id("test_products_only_flag")).click(); //Filter results
driver.sleep(100);
// Error: Element is not currently visible
driver.findElement(By.xpath("//table[@id='grid']/tbody/tr[1]/td/button")).click();

```

The last test statement would fail with an error “*Element is not currently visible*”. After checking the “Test automation products only” checkbox, we see only 2 rows on screen. However, there are still 4 rows in the page, the other two are hidden.

```

▼ <tbody>
  ▶ <tr class="service_products" style="display: none;">...</tr> == $0
  ▶ <tr>...</tr>
  ▶ <tr class="service_products" style="display: none;">...</tr>
  ▶ <tr>...</tr>
</tbody>

```

The button identified by this XPath `//table[@id='grid']/tbody/tr[1]/td/button` is now a hidden one, therefore unable to click.

A solution is to extract the visible rows to an array, then we could check them by index.

```

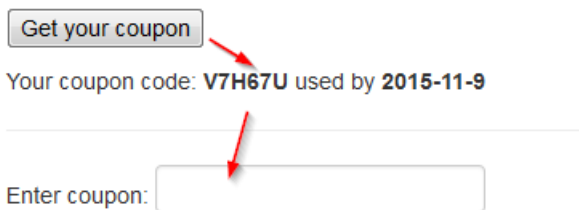
driver.findElements(By.xpath("//table[@id='grid']/tbody/tr[not(contains(@style,'display:\none'))]")).then(function(displayRows) {
  assert.equal(2, displayRows.length);
  var firstDisplayRow = displayRows[0]
  firstDisplayRow.findElement(By.xpath("td[1]")).getText().then(
    function(newFirstProductName) {
      assert.equal("BuildWise", newFirstProductName);
    });
  firstDisplayRow.findElement(By.xpath("td/button")).click();
})

```

20.11 Extract dynamic text following a pattern using Regex

To use dynamic data created from the application, e.g. receipt number, we need to extract them out. Ideally, those data are marked by dedicated IDs such as ``. However, it is not always the case, i.e., the data are mixed with other text.

The most commonly used approach (in programming) is to extract data with Regular Expression. Regular Expression (abbreviated *regex* or *regexp*) is a pattern of characters that finds matching text. Almost every programming language supports regular expression, with minor differences.



The test script below will extract “V7H67U” and “2015-11-9” from the text Your coupon code: V7H67U used by 2015-11-9, and enter the extracted coupon code in the text box.

```
driver.get("file://" + __dirname + "../..//site/coupon.html");
driver.findElement(By.id("get_coupon_btn")).click();
driver.findElement(By.id("details")).getText().then(function(couponText){
    console.log(couponText);
    var regex = /Your coupon code:\s+(\w+) used by\s([\d|-]+)/;
    var matchArray = regex.exec(couponText);
    if (matchArray.length > 1) {
        var coupon = matchArray[1]
        var expiryDate = matchArray[2]
        assert.equal(20, coupon.length)
        driver.findElement(By.name("coupon")).sendKeys(coupon);
    } else {
        throw("Error: no valid coupon returned")
    }
})
```

Regular expression is very powerful and it does take some time to master it. To get it going for simple text matching, however, is not hard. Google ‘java regular expression’ shall return some good tutorials, and [Rubular](http://rubular.com/)³ is a helpful tool to let you try out regular expression online.

³<http://rubular.com/>

21. Optimization

Working test scripts is just the first test step to successful test automation. As automated tests are executed often, and we all know the application changes frequently too. Therefore, it is important that we need our test scripts to be

- Fast
- Easy to read
- Concise

In this chapter, I will show some examples to optimize test scripts.

21.1 Assert text in page_source is faster than the text

To verify a piece of text on a web page, frequently for assertion, we can use `driver.getPageSource()` or `driver.findElement(By.tagName("body")).getText()`. Besides the obvious different output, there are big performance differences too. To get a text view (for a whole page or a web control), WebDriver needs to analyze the raw HTML to generate the text view, and it takes time. We usually do not notice that time when the raw HTML is small. However, for a large web page like the [WebDriver standard](http://www.w3.org/TR/webdriver/)¹ (over 430KB in file size), incorrect use of ‘text view’ will slow your test execution significantly.

¹<http://www.w3.org/TR/webdriver/>


```
driver.get("file://" + __dirname + "../../../site/WebDriverStandard.html");
var startTime = new Date().getTime();
driver.findElement(By.tagName("body")).getText().then(function(pageText) {
    var endTime = new Date().getTime();
    var duration = endTime - startTime;
    console.log("Method 1: Search whole document text took " + duration + " ms")
});

startTime = new Date().getTime();
driver.getPageSource().then(function(pageSource) {
    var endTime = new Date().getTime();
    var duration = endTime - startTime;
    console.log("Method 2: Search whole document html took " + duration + " ms")
});
```

Let's see the difference.

```
Method 1: Search whole document text took 3006 ms
Method 2: Search whole document html took    77 ms
```

21.2 Getting text from more specific element is faster

A rule of thumb is that we save execution time by narrowing down a more specific control. The two assertion statements largely achieve the same purpose but with big difference in execution time.

```
driver.findElement(By.tagName("body")).getText().then(function(pageText) {
    assert(pageText.contains("platform- and language-neutral wire protocol"))
});
```

Execution time: 3.121 seconds

```
driver.findElement(By.id("abstract")).getText().then(function(elemText) {
    assert(elemText.contains("platform- and language-neutral wire protocol"))
});
```

Execution time: 0.067 seconds

21.3 Avoid programming if-else block code if possible

It is common that programmers write test scripts in a similar way as coding applications, while I cannot say it is wrong. For me, I prefer simple, concise and easy to read test scripts. Whenever possible, I prefer one line of test statement matching one user operation. This can be quite helpful when debugging test scripts. For example, By using ternary operator `?` `:`, the below 7 lines of test statements

```
driver.findElement(By.id("notes")).getText().then(function(notesText){
    if (refNo.contains("VIP")) { // Special
        assert.equal("Please go upstairs", notesText)
    } else {
        assert.equal("", notesText)
    }
});
```

is reduced to three.

```
driver.findElement(By.id("notes")).getText().then(function(notesText){
    assert.equal(refNo.contains("VIP") ? "Please go upstairs" : "", notesText)
});
```

21.4 Enter large text into a text box

We commonly use `send_keys` to enter text into a text box. When the text string you want to enter is quite large, e.g. thousands of characters, try to avoid using `send_keys`, as it is not efficient. Here is an example.

```
var longStr = "0".repeat(1024 * 5)
var textAreaElem = driver.findElement(By.id("comments"))
textAreaElem.sendKeys(longStr);
```

Execution time: 11.642 seconds.

When this test is executed in Chrome, you can see a batch of text ‘typed’ into the text box. Furthermore, there might be a limited number of characters that WebDriver ‘send’ into a text box for browsers at one time. I have seen test scripts that broke long text into trunks and then sent them one by one, not elegant.

The **solution** is actually quite simple: using JavaScript.

```
// longStr is the same
driver.executeScript("document.getElementById('comments').value=arguments[0];", longStr);
```

Execution time: 0.061 seconds

21.5 Use Environment Variables to change test behaviours dynamically

Typically, there are more than one test environment we need to run automated tests against, and we might want to run the same test in different browsers now and then. I saw the test scripts like the below often in projects.

```
// declare global variables with default values
var targetBrowser = "firefox"
// var targetBrowser = "chrome"
var targetSiteUrl = "https://physio.clinicwise.net"
// var targetSiteUrl = "http://demo.poolwise.net";

driver = new webdriver.Builder().forBrowser(targetBrowser).build();
driver.get(targetSiteUrl)
```

It works like this: testers comment and uncomment a set of test statements to let test script run against different servers in different browsers. This is not an ideal approach, because it is inefficient, error prone and introducing unnecessary check-ins (changing test script files with no changes to testing logic).

A simple solution is to use agreed environment variables, so that the target server URL and browser type can be set externally, outside the test scripts.

```
// global variables
var targetBrowser = "firefox"
var targetSiteUrl = "https://physio.clinicwise.net"

// ...

test.it("Use Environment variables to change test behaviour dynamically", function() {

  if (process.env.TARGET_SITE_URL) {
    targetSiteUrl = process.env.TARGET_SITE_URL
  }
  if (process.env.TARGET_BROWSER) {
```

```

        targetBrowser = process.env.TARGET_BROWSER
    }
    driver = new webdriver.Builder()
        .forBrowser(targetBrowser)
        .build();
    driver.get(targetSiteUrl)
});

```

For example, to run this test against another server in Chrome, run below commands.

```

> set TARGET_BROWSER=chrome
> set SITE_URL=http://yake.clinicwise.net

```

This approach is commonly used in Continuous Testing process.

21.6 Testing web site in two languages

The test scripts below to test user authentication for two test sites, the same application in two languages: *http://physio.clinicwise.net* in English and *http://yake.clinicwise.net* in Chinese. While the business features are the same, the text shown on two sites are different, so are the test user accounts.

```

driver.get(targetSiteUrl); // may be dynamically set by env variable

if (targetSiteUrl.contains("physio")) {
    driver.findElement(By.id("username")).sendKeys("natalie");
    driver.findElement(By.id("password")).sendKeys("test");
    driver.findElement(By.id("signin_button")).click();
    driver.getPageSource().then(function(pageSource){
        assert(pageSource.contains("Signed in successfully. "))
    });
} else if (targetSiteUrl.contains("yake")) {
    driver.findElement(By.id("username")).sendKeys("tuo");
    driver.findElement(By.id("password")).sendKeys("test");
    driver.findElement(By.id("signin_button")).click();
    driver.getPageSource().then(function(pageSource){
        assert(pageSource.contains("你已成功登录"))
    });
}

```

Though the above test scripts work, it seems lengthy and repetitive.

```

function isChinese() {
  return targetSiteUrl.contains("yake");
}

test.it("Two language testing: with TernaryOperator", function() {
  // ...
  driver.get(targetSiteUrl); // may be dynamically set by env variable

  driver.findElement(By.id("username")).sendKeys(isChinese() ? "tuo" : "natalie");
  driver.findElement(By.id("password")).sendKeys("test");
  driver.findElement(By.id("signin_button")).click();
  driver.getPageSource().then(function(pageSource){
    assert(pageSource.contains(isChinese() ? "你登录成功" : "Signed in successfully. "))
  });
});

```



Using IDs can greatly save multi-language testing

When doing multi-language testing, try not to use the actual text on the page for non user-entering operations. For example, the test statements are not optimal.

```

driver.findElement(By.linkText("Register")).click();
// or below with some programming logic ...
driver.findElement(By.linkText("Registre")).click(); // french
driver.findElement(By.linkText("你注册")).click();    // chinese

```

Using IDs is much simpler.

```
driver.findElement(By.id("register_link")).click();
```

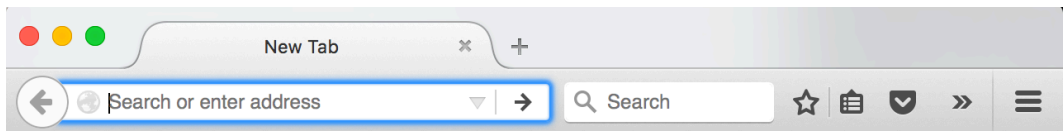
This works for all languages.

22. Gotchas

For the most part, Selenium WebDriver API is quite straightforward. My one sentence summary: find a element and perform an operation on it. Writing test scripts in Selenium WebDriver is much more than knowing the API, it involves programming, HTML, JavaScript and web browsers. There are cases that can be confusing to newcomers.

22.1 Test starts browser but no execution with blank screen

A very possible cause is that the version of installed Selenium WebDriver is not compatible with the version of your browser. Here is a screenshot of Firefox 41.0.2 started by a Selenium WebDriver 2.44.0 test.



The test hung there. After I upgraded Selenium WebDriver to 2.45, the test ran fine.

This can happen to Chrome too. With both browsers and Selenium WebDriver get updated quite frequently, in a matter of months, it is not that surprising to get the incompatibility issues. For test engineers who are not aware of this, it can be quite confusing as the tests might be running fine the day before and no changes have been made since.

Once knowing the cause, the solutions are easy:

- Upgrade both Selenium WebDriver and browsers to the latest version

Browsers such as Chrome usually turn on auto-upgrade by default, I suggest upgrading to the latest Selenium WebDriver several days after it is released.

- Lock Selenium WebDriver and browsers.

Turn off auto-upgrade in browser and be thoughtful on upgrading Selenium Webdriver.



Be aware of browser and driver changes

One day I found over 40 test failures (out of about 400) by surprise on the latest continuous testing build. There were little changes since the last build, in which all tests passed. I quickly figured out the cause: Chrome auto-upgraded to v44. Chrome 44 with the ChromeDriver 2.17 changed the behaviour of clicking hyperlinks. After clicking a link, sometimes test executions immediately continue to the next operation without waiting for the “clicking link” operation to finish.

```
driver.findElement(By.id("new_client")).click();  
// workaround for chrome v44, make sure the link is clicked  
driver.sleep(1000);
```

A week later, I noticed the only line in the change log of ChromeDriver v2.18:

```
"Changes include many bug fixes that allow ChromeDriver to work  
more reliably with Chrome 44+."
```

22.2 Failed to assert copied text in browser

To answer this, let's start with an example. What we see in a browser (Internet Explorer)

```
BOLD Italic  
Text assertion  
(new line before)!
```

is the result of rendering the page source (HTML) below in Internet Explorer:

```
<p id="text"> <b>BOLD</b> <i>Italic</i> </p>  
<pre id="formatted">Text assertion &nbsp;   </pre>  
(new line before)!</pre>
```

As you can see, there are differences. Test scripts can be written to check the text view (what we saw) on browsers or its raw page source (HTML). To complicate things a little more, old versions of browsers may return slightly different results.

Don't worry. As long as you understand the text shown in browsers are coming from raw HTML source. After a few attempts, this is usually not a problem. Here are the test scripts for checking text and source for above example:

```
driver.get("http://testwisely.com/demo/assertion")
driver.findElement(By.tagName("body")).getText().then(function(pageText) {
    assert(pageText.contains("BOLD Italic"))
})
driver.getPageSource().then(function(pageSource) {
    assert(pageSource.contains("<b>BOLD</b>  <i>Italic</i>"))
})

// HTML entities in source but shown as space in text
driver.findElement(By.tagName("body")).getText().then(function(pageText) {
    assert(pageText.contains("assertion \n(new line before)"))
});

driver.findElement(By.id("formatted")).getAttribute("innerHTML").then(function(innerHtml)\
{
    assert(innerHtml.contains("assertion &nbsp;\n(new line before)"))
});

// Note the second character after text 'assertion' is not a space character
driver.getPageSource().then(function(pageSource) {
    assert(pageSource.contains("assertion \n(new line before)"))
})
```

22.3 The same test works for Chrome, but not IE

Chrome, Firefox and IE are different products and web browsers are very complex software. Comparing to other testing frameworks, Selenium WebDriver provides better support for all major browsers. Still there will be some operations work differently on one than another.


```

driver.getCapabilities().then(function(caps){
  var browserName = caps.get("browserName");
  if (browserName == "firefox") {
    // firefox specific test statement
  } else if (browserName == "chrome") {
    // chrome specific test statement
  } else {
    throw "unsupported browser: " + browserName;
  }
});

```

Some might say that it will require a lot of work. Yes, cross-browser testing is associated with more testing effort, obviously. However, from my observation, few IT managers acknowledge this. That's why cross-testing is talked a lot, but rarely gets done.

22.4 “unexpected tag name ‘input’”

This is because there is another control matching your `findElement` and it is a different control type (input tag). For example,

```

<input type="checkbox" name="vip" value="on"> VIP?

<!-- ... -->
<select name="vip">
  <option value="true">Yes</option>
  <option value="false">No</option>
</select>

```

The intention of the test script below's intention is to select 'Yes' in the dropdown list, but not aware of there is another checkbox control sharing exactly the same name attribute.

```

driver.get("file://" + __dirname + "../site/gotchas.html");

var selElem = driver.findElement(By.name("vip"));
selElem.findElement(By.css("option:nth-child(2)")).click();

```

The above script will return error reporting “Unable to locate element: {“method”:“css selector”,“selector”:“option:nth-child(2)”}”. In other language bindings, calling Select specific operation `selectByVisibleText` will get this error.

UnexpectedTagNameException: Element should have been "select" but was "input"

The reason: there are two controls with the same “name” attribute “vip”. `By.name("vip")` returns the first matching control, i.e., the text field. The solution is quite obvious after knowing the cause: change the locator to be more specific `By.xpath("//select[@name='vip']")`.

A quite common scenario is as below: a hidden element and a checkbox element share the same ID and NAME attributes.

```
<input type="hidden" name="vip" value="false"/>
<!-- ... -->
<input type="checkbox" name="vip" value="on"> VIP?
```

In this case, there might be no error thrown. However, this can be more subtle, as the operation is applied to a different control.

22.5 Element is not clickable or not visible

Some controls such as text fields, even when they are not visible in the current browser window, Selenium WebDriver will move the focus to them. Some other controls such as buttons, may be not. In that case, though the element is found by `findElement`, it is not clickable.

The solution is to make the target control visible in browser.

1. Scroll the window to make the control visible

Find out the control’s position and scroll to it.

```
elem = driver.findElement(By.name("submit_action_2"));
elemPos = elem.getLocation().y
driver.executeScript("window.scroll(0, " + elemPos + ");"); // scroll
```

Or scroll to the top / bottom of page.

```
driver.executeScript("window.scrollTo(0, document.body.scrollHeight);");
```

2. A hack, call `sendKeys` to a text field nearby, if there is one.

23. Material Design Web App

Material Design (MD) is a design language created in 2014 by Google, has become huge popular for web apps. In the chapter I use Materialize (a popular Material Design framework) as an example to show how to test Material Design controls with Selenium WebDriver.

23.1 Select List (dropdown)

Default Materialize Select style.

```
driver.get("https://whenwise.agileway.net");
elem_id = "businessType";
option_label = "Physio"
driver.findElement(By.xpath("//select[@id='" + elem_id + "']/..")).click();
driver.sleep(500);
driver.findElement(By.xpath("//select[@id='" + elem_id + "']/../ul/li/span[text()=' " +
    option_label + "']")).click();
driver.sleep(500);
```

23.2 Checkbox

Toggle a checkbox.

```
driver.get("https://whenwise.agileway.net/sign-in");
// the below won't work
// driver.findElement(By.id("remember_me")).click();

// this works
driver.findElement(By.xpath("//input[@id='remember_me']/../span")).click();
```

Check a checkbox.

```
if (!driver.findElement(By.id( "remember_me")).isSelected()) {  
    driver.findElement(By.xpath("//input[@id='remember_me']/../span")).click();  
}
```

Uncheck a checkbox.

```
if (driver.findElement(By.id( "remember_me")).isSelected()) {  
    driver.findElement(By.xpath("//input[@id='remember_me']/../span")).click();  
}
```

23.3 Drag range (noUiSlider)

noUiSlider is a Materialize plugin for Range control.

```
driver.get("https://whenwise.agileway.net/biz/wise-driving");  
driver.sleep(1000);  
driver.findElement(By.id("slider-start-time")).getText().then(function(the_time) {  
    assert.equal("09:00", the_time);  
});  
elem = driver.findElement(By.className("noUi-handle-lower"));  
// advance one hour, the offset value of 60 is try-then-find-out  
driver.actions().dragAndDrop(elem, {x: 60, y: 0}).perform();  
driver.sleep(500);  
driver.findElement(By.id("slider-start-time")).getText().then(function(the_time) {  
    assert.equal("10:00", the_time);  
});
```

23.4 Verify Toast message

Toast is Materialize's way to send user alerts.

```
this.timeout(timeOut);
var faker = require('faker');
driver.get("https://whenwise.agileway.net/sign-up");
driver.findElement(By.id("email")).sendKeys(faker.internet.email());
driver.findElement(By.id("password")).sendKeys("test01");
driver.findElement(By.id("create-account")).click();
// Materialize toast fade in a few seconds
driver.sleep(1000);
driver.findElement(By.id("toast-container")).getText().then(function(toast_text){
    assert(toast_text.contains("Please check your email to activate your account"));
});
```

23.5 Modal

```
driver.get("https://whenwise.agileway.net/biz/wise-driving");
driver.sleep(1000);
driver.findElement(By.id("reviews-link")).click();
// wait for modal to show up
ok_btn_xpath = "//*[@id='modal-reviews']/a[@id='review-modal-ok']"
driver.wait(until.elementLocated(By.xpath(ok_btn_xpath)), 4 * 1000);
```

Verify text in a modal.

```
driver.findElement(By.id("modal-reviews")).getText().then(function(the_modal_text) {
    assert(the_modal_text.contains("It has been a pleasure to"));
});
```

24. Selenium Remote Control Server

Selenium Server, formerly known as Selenium Remote Control (RC) Server, allows testers to write Selenium tests in their favourite language and execute them on another machine. The word 'remote' means that the test scripts and the target browser may not be on the same machine.

The Selenium Server is composed of two parts: a server and a client.

- **Selenium Server.** A Java server which launches, drives and kills browsers on receiving commands, with the target browser installed on the machine.
- **Client libraries.** Test scripts in tests' favourite language bindings, such as Ruby, Java and Python.

24.1 Selenium Server Installation

Make sure you have Java Runtime installed first. Download Selenium Server *selenium-server-standalone-{VERSION}.jar* from [Selenium download page](http://www.seleniumhq.org/download/)¹ and place it on the computer with the browser(s) you want to test. Then from the directory with the jar run the following the Command Line

```
java -jar selenium-server-standalone-2.53.1.jar
```

Sample output

¹<http://www.seleniumhq.org/download/>

```
09:34:41.616 INFO - Launching a standalone Selenium Server
09:34:41.747 INFO - Java: Oracle Corporation 25.77-b03
09:34:41.747 INFO - OS: Mac OS X 10.11.5 x86_64
09:34:41.778 INFO - v2.53.1, with Core v2.53.1. Built from revision a36b8b1
...
09:37:43.973 INFO - RemoteWebDriver instances should connect to: http://127.0.0.1:4444/wd\
/hub
09:37:43.973 INFO - Selenium Server is up and running
```

There are two options you can pass to the server: `timeout` and `browserTimeout`.

```
java -jar selenium-server-standalone-2.53.1.jar -timeout=20 -browserTimeout=60
```

24.2 Execute tests in specified browser on another machine

Perquisite:

- Make sure the Selenium Server is up and running.
- You can connect to the server via HTTP.
- Note down the server machine's IP address.

To change existing local Selenium tests (running on a local browser) to remote Selenium tests (running on a remote browser) is very easy, just update the initialization of WebDriver instance to using `usingServer` like below:

```
test.it("Remote control Firefox on another machine", function() {
    this.timeout(timeOut);
    driver = new webdriver.Builder()
        .usingServer('http://localhost:4444/wd/hub')
        .forBrowser('firefox')
        .build();
    driver.get("http://travel.agileway.net")
    driver.findElement(webdriver.By.name('username')).sendKeys('agileway');
    driver.findElement(webdriver.By.name('password')).sendKeys('testwise');
    driver.findElement(webdriver.By.name('commit')).click();
    driver.quit();
});
```

The test scripts (client) is expected to terminate each browser session properly, calling `driver.quit`.

24.3 Selenium Grid

Selenium Grid allows you to run Selenium tests in parallel to cut down the execution time. Selenium Grid includes one hub and many nodes.

1. Start the Hub

The hub receives the test requests and distributes them to the nodes.

```
java -jar selenium-server-standalone-2.53.1.jar -role hub
```

2. Start the nodes

A node gets tests from the hub and run them.

```
java -jar selenium-server-standalone-2.53.1.jar -role node \
    -hub http://localhost:4444/grid/register
```

If you starts a node on another machine, replace *localhost* with the hub's IP address.

3. Using grid to run tests

You need to change the test script to point to the driver to the hub.

```
driver = new webdriver.Builder()
    .usingServer('http://localhost:4444/wd/hub')
    .forBrowser('chrome')
    .build();
```

The test will run on one of the nodes. Please note that the timing and test case counts (from RSpec) returned is apparently not right.

Frankly, I haven't yet met anyone who is able to show me a working selenium-grid running a fair number of UI selenium tests.

Here are my concerns with Selenium Grid:

- **Complexity**

For every selenium grid node, you need to configure the node either by specifying command line parameters or a JSON file. Check out the [Grid Wiki page](https://code.google.com/p/selenium/wiki/Grid2)² for details.

It is my understanding that just pushing the tests to the hub, and it handles the rest based on the configuration. My experience tells me that it is too good to be true. For example, here is an error I got. While the error message is quite clear: no ChromeDriver installed. But on which node? Shouldn't the hub 'know' about that?

²<https://code.google.com/p/selenium/wiki/Grid2>


```
[remote server] com.google.common.base.Preconditions(Preconditions.java):177:in
`checkState': The path to the driver executable must be set by the
webdriver.chrome.driver system property; for more information,
see http://code.google.com/p/selenium/wiki/ChromeDriver.
(java.lang.IllegalStateException) (Selenium::WebDriver::Error::UnknownError)
```

- **Very limited control**

Selenium-Grid comes with a web accessible console, in my view, very basic one. For instance, I created 2 nodes: one on Mac; the other on Windows 7 (the console displayed as 'VISTA').



An IE icon for for Mac node? This does not seem right.

- **Lack of feedback**

UI tests take time to execute, more tests means longer execution time. Selenium Grid's distribution model is to reduce that. Apart from the raw execution time, there is also the feedback time. The team would like to see the test results as soon as a test execution finishes on one node. Even better, when we pass the whole test suite to the hub, it will 'intelligently' run new or last failed tests first. Selenium Grid, in my view, falls short on this.

- **Lack of rerun**

In a perfect world, all tests execute as expected every single time. But in reality, there are so many factors that could affect the test execution:

- test statements didn't wait long enough for AJAX requests to complete (server on load)
- browser crashes (it happens)

- node runs out of disk space
- virus scanning process started in background
- windows self-installed an update
- ...

In this case, re-assign failed tests to another node could save a potential good build.

My point is: I could quickly put together a demo with Selenium Grid running tests on different nodes (with different browsers), and the audience might be quite impressed. However, in reality, when you have a large number of UI test suites, the game is totally different. The whole process needs to be simple, stable, flexible and very importantly, being able to provide feedback quickly. In the true spirit of Agile, if there are tests failing, no code shall be allowed to check in. Now we are talking about the pressure ...

How to achieve distributed test execution over multiple browsers? First of all, distributed test execution and cross browser testing are two different things. Distributed test execution speeds up test execution (could be just against single type of browser); while cross-browser testing is to verify the application's ability to work on a range of browsers. Yes, distributed test execution can be used to test against different browsers. But do get distributed test execution done solidly before worrying about the cross browser testing.

I firmly believe the UI test execution with feedback shall be a part of continuous integration (CI) process, just like running xUnit tests and the report shown on the CI server. It is OK for developers/testers to develop selenium tests in an IDE, in which they run one or a handful tests often. However, executing a large number of UI tests, which is time consuming, shall be done in the CI server.

The purpose of a perfect CI process: building the application to pass all tests, to be ready to production release. Distributed execution of UI tests with quick feedback, in my opinion, is an important feature of a CI Server. However, most CI servers in the market do not support this feature. You can find more information on this topic in my other book *Practical Web Test Automation*³.

³<https://leanpub.com/practical-web-test-automation>

25. Quiz

There is maybe more than one correct answer for multiple-choice questions.

1. Introduction

1.1 Which of the following browsers does Selenium WebDriver officially support?

- (A) Chrome (B) Firefox (C) Edge v84+ (D) Safari (E) All of the above

1.2 Which of the following are BDD frameworks?

- (A) JUnit (B) RSpec (C) Cucumber (D) All of the above

1.3 Which of the following is NOT an official Selenium WebDriver language binding?

- (A) Java (B) Ruby (C) Go (D) Python (E) C#

1.4 Which of the following software are required for running a Selenium script in a Chrome browser.

- (A) Programming language runtime, e.g. Ruby
(B) Selenium library, e.g. `selenium-webdriver` library
(C) Browser, e.g. Google Chrome
(D) Browser driver (e.g. `ChromeDriver`) matches the browser's version and it is in the `PATH`
(E) All of the above

2. Locators

2.1 Generally speaking, which Selenium WebDriver locator is the fastest?

- (A) ID (B) Name (C) Xpath (D) Class name (E) CSS selector

2.2 Generally speaking, which Selenium WebDriver locator is the most flexible?

- (A) ID (B) Name (C) Xpath (D) Class name (E) CSS selector

2.3 **True or false** – Selenium supports finding child elements with locator chaining, e.g.

```
driver.findElement(By.id("div2")).findElement(By.name("same")).click();
```

3. Hyperlink

3.1 **True or false** – The TEXT in the Link text locator `driver.findElement(By.linkText("TEXT"))` is case insensitive.

3.2 If there are two hyperlinks with exact the same text “TestWise”, what will this test step `By.linkText("TestWise")` do?

- (A) Click the first (B) Click the last (C) Neither (D) Indeterminately

3.3 **True or false** – Selenium can click the link text in Unicode, e.g. Japanese.

3.4 Which of the following Selenium test step clicks the second link with the same text?

(A) `driver.findElements(By.linkText("Same")).then(function(the_same_links) { the_same_links[2].click(); });`

(B) `driver.findElements(By.linkText("Same")).then(function(the_same_links) { the_same_links[1].click(); });`

4. Button

4.1 **True or false** – It is OK to click a disabled button in Selenium, just no effect.

4.2 If there are buttons on a web page, `driver.findElement(By.css("form#login > input:nth-child(2)")).click()` will click which button?

- (A) the first button (B) the second button (C) the third button

4.3 Which are the ways to submit a form on a web page?

(A) Click a submit button, `driver.findElement(By.id("sign-in")).click();`

(B) Call submit on a form element, `driver.findElement(By.id("password")).submit();`

(C) Send “Enter” key to a submit button,

`driver.findElement(By.id("sign-in")).sendKeys(webdriver.Key.ENTER);`

(D) All of the above

5. Text Field

5.1 What will be text in the text field (#user with no text initially) after executing the following Selenium test steps.

```
driver.findElement(By.id("user")).sendKeys("foo");  
driver.findElement(By.id("user")).sendKeys("bar");
```

- (A) foo (B) bar (C) foobar

5.2 **True or false** – Selenium can sendKeys to a read only-text field.

5.3 **True or false** – Selenium treats a text field, number field and textarea the same when sending texts to them.

5.4 #pass is a text field on a web page. What `driver.findElement(By.id("pass")).sendKeys("")` do?

- (A) Type one space in the text box (B) Focus on the text box
(C) Does nothing (D) Invalid

5.5 #comments is a textarea control on a web page. What text will be in textarea after running the scripts below?

```
elem = driver.findElement(By.id("comments"));  
elem.sendKeys("TestWise", " ", "is", webdriver.Key.ENTER);  
elem.sendKeys("Cooo", webdriver.Key.BACK_SPACE, "\n");
```

- (A) Invalid! (B) "TestWise is :enter Cooo:backspacel"
(C) "TestWise is" (new line) "Cool" (D) "Cool"

6. Radio button

6.1 What is the result of running this Selenium step

```
driver.findElement(By.id("radio_female")).clear();
```

- (A) Error, the clear method is not available for this
(B) OK, the radio option is cleared
(C) OK, no effect
(D) Error, unable to perform clear on this element

7. CheckBox

7.1 For a checkbox control `elem = driver.findElement(By.id("accept"))`, which of the following checks it safely?

- (A) `elem.click`
- (B) `the_checkbox.isSelected().then(function(selected) { if (!selected) { the_checkbox.click(); } });`
- (C) `elem.clear(); elem.click();`

7.2 For a checkbox control `elem = driver.findElement(By.id("accept"))`, which of the following unchecks it safely?

- (A) `elem.clear()`
- (B) `elem.click()`
- (C) `the_checkbox.isSelected().then(function(selected) { if (selected) { the_checkbox.click(); } });`

8. Select list

8.1 True or false – Selenium can `sendKeys` to select an option in a Select list.

9. Navigation and Browser

9.1 Which of the following test steps are correct for opening a web page in Selenium?

- (A) `driver.get("whenwise.com")`
- (B) `driver.navigate.to("https://whenwise.com")`
- (C) `driver.get("https://whenwise.com")`

9.2 For a browser with multiple tabs, what will the below script do?

```
driver.getAllWindowHandles().then(function(allWindows){
    driver.switchTo().window(allWindows[allWindows.length - 1])
});
```

- (A) Error, this step is invalid
- (B) Change to the previous tab
- (C) Change to the last tab

9.3 Which of the following are correct about `driver.quit()` and `driver.close()`?

- (A) `driver.quit()` and `driver.close()` work the same.
- (B) `driver.quit()` closes the whole browser, `driver.close()` closes the current tab
- (C) `driver.close()` closes the whole browser, `driver.quit()` closes the current tab
- (D) `driver.close()` and `driver.quit()` are equivalent if there is only one tab

10. Assertion

10.1 **True or false** – assertion syntaxes are provided by the test syntax framework, not the automation framework such as Selenium WebDriver.

11. Frame

11.1 **True or false** – Selenium treats normal frames and iframes the same way.

11.2 Which of the following are correct to switch to a frame?

- (A) `driver.switchTo().frame("frame_name")`
- (B) `driver.switchTo().frame("frame_id")`
- (C) `elem = driver.findElement(By.id("frame_id")); driver.switchTo().frame(elem)`
- (D) All of the above

12. AJAX

12.1 What are the ways to test an AJAX operation in Selenium?

- (A) Wait for a fixed time, e.g. `driver.sleep(5000)`
- (B) Explicit waits until times out
- (C) Implicit Waits until times out
- (D) Use programming code to do polling checks
- (E) All of the above

13. Popup

13.1 **True or false** – A test execution can continue with a JavaScript alert popup.

13.2 **True or false** – To upload a file using Selenium, find the `input[type=file]` element, send the file path to it by using `sendKeys`.

14. Debugging

14.1 **True or false** – Code debuggers in programming IDEs are inefficient at debugging automated test scripts.

14.2 Which of the followings are essential techniques for debugging automated test scripts for web apps?

- (A) Keep the browser open (for inspection)
- (B) Run selected test steps against the current browser
- (C) Error stack trace
- (D) Save a screenshot of the web page when a test failure occurs
- (E) All of the above

15. Test Data

15.1 **True or false** – Test data is not a part of the test script.

15.2 **True or false** – Programming makes generating test data much more flexible.

16. Browser profile

16.1 **True or false** – Headless mode can be enabled by creating a WebDriver session with a custom profile.

16.2 **True or false** – We can set the Downloads folder of the browser that Selenium starts.

17. Advanced User Interactions

17.1 **True or false** – “Drag and drop” is not possible with Selenium.

17.2 Assume `elem` is a text field with some text on a web page on a Mac computer. Which following statements delete all text in it?

- (A) `elem.clear()`
- (B) `elem.sendKeys(Key.COMMAND, "a"); elem.sendKeys(Key.BACK_SPACE)`
- (C) `driver.actions().click(elem).keyDown(Key.COMMAND).sendKeys("a").keyUp(Key.COMMAND).sendKeys(Key.BACK_SPACE).perform();`
- (D) All of the above

18. JavaScript

18.1 **True or false** – We can use JavaScript in Selenium scripts to scroll the browser to a specific web control.

18.2 How to trigger an 'OnChange' event for a textfield #login in Selenium?

- (A) `driver.executeScript("$('#login').trigger('change')")`
- (B) `driver.executeScript("document.getElementById('b1').fireEvent('OnChange')")`
- (C) `driver.executeScript("var elem = document.getElementById('person_name_textbox');
var event = new Event('change'); elem.dispatchEvent(event);
")`
- (D) All of the above

25.1 Answers

1.1 (E)

Selenium Webdriver also supports IE6+ and Opera 10.5 as well, [the full list here](#)¹.

1.2 (B) (C)

JUnit is a unit test framework.

1.3 (C)

1.4 (E)

2.1 (A)

2.2 (C)

2.3 (A)

3.1 False

3.2 (A)

3.3 True

3.4 (B)

The index in JavaScript (and pretty much all programming languages) starts from 0.

4.1 True

4.2 (B)

The `:nth-child()` in CSS starts from 1.

4.3 (D)

5.1 (C)

5.2 False

However, it is possible using Selenium `executeScript` to set the value of a read-only text field.

5.3 True

5.4 (B)

5.5 (C)

¹https://www.selenium.dev/documentation/en/getting_started_with_webdriver/browsers/

6.1 **(D)**

7.1 **(B)**

(C) is invalid, there is no `clear()` function for a checkbox element in Selenium.

7.2 **(C)**

8.1 **True**

9.1 **(B) (C)**

9.2 **(B)**

Note, some browsers might behave differently on this.

9.3 **(B) (D)**

10.1 **True**

11.1 **True**

11.2 **(D)**

12.1 **(E)**

13.1 **False**

13.2 **True**

14.1 **True**

14.2 **(E)**

15.1 **False**

Test data is in the test script and being used, of course, it is the part of test scripts. Using test data as excuses for failed automated test scripts is wrong.

15.2 **True**

With programming, we can generate random data (within a set), load from a CSV/Excel file or retrieve from the database using SQL.

16.1 **True**

The syntax for setting headless mode varies for different browsers.

16.2 **True**

It is a good practice to set the Downloads folder before testing file downloads.

17.1 **False**

```
driver.actions().dragAndDrop(dragFrom, target).perform();
```

17.2 (D)

18.1 True

```
elem = driver.findElement(By.name("agree_btn"));  
elem_pos = elem.getLocation().y  
driver.executeScript("window.scroll(0, " + elem_pos + ");")
```

18.2 (A C)

fireEvent (in B) no longer works in modern browsers.

Appendix - Continuous Testing

This book shows recipes on solving individual test scenarios with Selenium WebDriver. Ideally, we run all the test scripts to verify every new build as regression testing. However, when total execution time of all tests exceeds one hour, it is becoming impractical to run them often on testers' machines. The solution is to execute tests on a Continuous Testing server with following benefits:

- Easier to run
- View test reports and error screenshots
- Test history
- Less interruption

Continuous testing is a big topic, here I will just cover a bare minimum to show you one approach to run a suite of Selenium (in JavaScript) test scripts with a click of a button.

Verify server machine can run Selenium Mocha

Before we start running Selenium python tests in BuildWise server, it is a good idea to make sure we can run selenium tests from command line on this build server machine.

1. Install Node.js

Standard installation, see Chapter 1 for more.

2. Install Mocha, Selenium and required test modules

```
> npm install -g mocha
> npm install -g mocha-junit-reporter

> npm install -g selenium-webdriver@3.6.0
> npm install -g faker
```

3. Verify that you can run Selenium Mocha tests on the build server machine.

In the exercise, we will use a sample Git repository on [GitHub](https://github.com/testwisely/agiletravel-ui-tests)². The folder containing Selenium Python test scripts is selenium-webdriver-nodejs-mocha.

Start a new command window (Terminal on Mac/Linux) and run the following commands to verify test execution.

²<https://github.com/testwisely/agiletravel-ui-tests>

```
> cd temp
> git clone https://github.com/testwisely/agiletravel-ui-tests
> cd agiletravel-ui-tests/selenium-webdriver-nodejs-mocha
> npm install
> mocha spec/01_login_spec.js
```

Install BuildWise Server

1. Install Ruby

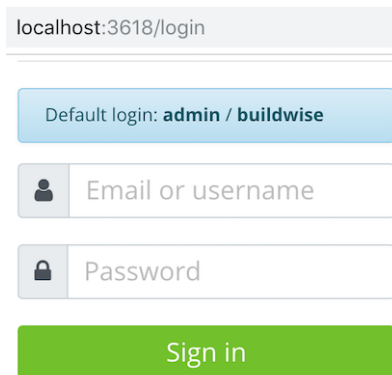
BuildWise requires Ruby to run. I recommend to download and install [RubyShell](#)³, a packaged installer of standard Ruby for Windows and all required libraries (called Gems in Ruby) for BuildWise.

2. Install BuildWise Server

Download and run BuildWise standalone Windows installer (BuildWiseStandalone-X-setup.exe) from [TestWisely](#)⁴. For Unix/Mac users, there is also a buildwise-X.zip. There will be a few more steps required for setting up Git, Ruby and Gems, though it is quite easy to find tutorials for these general knowledge online.


3. Start up BuildWise Server


Double click **startup.bat** under folder *C:\agileway\buildwise-standalone*, then open *http://localhost:3618* in your browser.



localhost:3618/login

Default login: **admin / buildwise**

 Email or username

 Password

Sign in

4. Login

The default login/password: *admin/buildwise*.

³<http://testwisely.com/testwise/downloads>

⁴<http://testwisely.com/buildwise/downloads>



Create Build Project in BuildWise

BuildWise uses a concept of ‘Project’. A quick way to create a project in BuildWise is providing a name (for display), an identifier, and a local working folder containing test scripts. To make it easy for beginners, BuildWise includes a set of sample project configurations (with code hosted on Github) for different test frameworks. In the tutorial, I select ‘Fill demo project’ → ‘Mocha (Node.js)’.

New project

Option 1. Loading from a working folder or [Specify manually](#)

Project name:

AgileTravel Quick Build Mocha

Identifier:

agiletravel-quick-build-mocha

(lower case and unique, e.g. clinicwise-ui-test)

Working folder:

/Users/zhimin/work/projects/agiletravel-ui-tests

(Specify the SCM checked out project folder on the machine running BuildWise server, e.g. /home/you/work. BuildWise will try to use the working git's origin URL as the repository URL, can be changed later along with the project working folder under ~/.buildwise).

UI test folder:

selenium-webdriver-nodejs-mocha/spec

(where the UI tests are, relative to project root directory, eg. spec or ui-tests/spec)

Test results folder:

selenium-webdriver-nodejs-mocha/reports

The directory (relative path) contains generated JUnit style test reports (TEST-XXX.xml), such as spec/reports or log. The output of test reports is set by Rakefile. This setting is for Quick Build only.

Rake task for UI or API Testing:

-f selenium-webdriver-nodejs-mocha/Rakefile ci:ui_tests:quick

(The task executing long running UI/API tests, e.g. ci:ui_tests:quick. You may configure this later.)

UI test framework:

Mocha (JavaScript)

Fill demo project

RSpec

Mocha (Node.js)

Cucumber

unittest (Python)

Create

Cancel

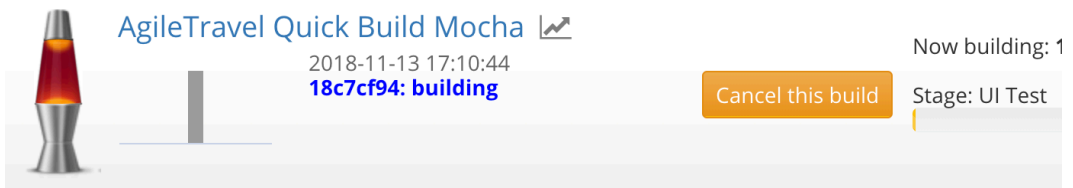
This will create the project *AgileTravel Quick Build Mocha* in BuildWise.

Trigger test execution manually

To start a build (for our purpose, 'Build' means execution of automated functional tests), click 'Build Now' button.



The colour of lava lamp (for the project) is now changed to orange, indicating a build is under way.




Soon you will see a browser window launching and your tests executing.

Feedback while test execution in progress


Click a build label (such as '1 :building') to show details of test execution:

buildwise so far so good admin ▾

AgileTravel Quick Build Mocha  Build #1 building ... ⋮ ▾

UI Test Started: 2018-11-13 17:57 Elapsed: 17 seconds

🕒 UI Test Results (2 tests) 🚩 Export ▾

🕒	TEST FILE (2 test cases)	TIME (S)	RESULT
11-13 17:57	01_login_spec.js 	9.2	
	- Invalid user	5.7	OK
	- User can login successfully	3.5	OK

You can see results of a test script as soon as it finishes execution, no need to wait for the whole suite to complete. The above screenshot was taken when the first test script (containing 2 test cases) finished execution.


You may inspect test failures (if any) on BuildWise.

04_payment_spec.js : '[5] Book flight with payment' ✕

Test output

```
NoSuchElementException: no such element: Unable to locate element: {"method":"xpath","selector":"//input[@value='Pay later']"}
(Session info: chrome=70.0.3538.102)
(Driver info: chromedriver=2.42.591059 (a3d9684d10d61aa0c45f6723b327283be1ebaad8),platform=Mac OS X 10.14.1 x86_64)
  at Object.checkLegacyResponse (selenium-webdriver-nodejs-mocha/node_modules/selenium-webdriver/lib/error.js:546:15)
  at parseHttpResponse (selenium-webdriver-nodejs-mocha/node_modules/selenium-webdriver/lib/http.js:509:13)
  at doSend.then.response (selenium-webdriver-nodejs-mocha/node_modules/selenium-webdriver/lib/http.js:441:30)
  at process.internalTickCallback (internal/process/next_tick.js:77:7)
From: Task: WebDriver.findElement(By(xpath, //input[@value='Pay later']))
From: Task: WebElement.click()
From: Task: Payment [5] Book flight with payment
```

Close Copy error line to Clipboard


11-13 17:57	04_payment_spec.js 	1.9	
	- [5] Book flight with payment	1.9	Failure

From the error or failure description, we can identify the cause. From line numbers in the error trace, we can easily navigate to the test case where the error occurred.

Please note that the tests are executed on the build server, not on your machine (unless you are running BuildWise locally). You can now open test cases in TestWise on your computer and run them, without having to wait for the CI build to complete.

Build finished

When a test execution completes, you will get the full test results shown on BuildWise. You can export them to an Excel spreadsheet. For the build below, only one out of five automated test cases failed.


1 failures
admin ▾

AgileTravel Quick Build Mocha
Build #1 Failed
Build Now


Started: 2018-11-13 17:10 Duration: 46 seconds

- Change log
- Build artifacts
- UI Test Results (5 tests)

Export ▾

	TEST FILE (5 test cases in 3 test scripts files)	TIME (S)	RESULT ^
11-13 17:11	04_payment_spec.js	15.0	
	- [5] Book flight with payment	15.0	Failure
11-13 17:10	01_login_spec.js	2.7	
	- Invalid user	0.7	OK
	- User can login successfully	2.0	OK
11-13 17:11	02_flight_spec.js	1.6	
	- [3] Return trip	0.9	OK
	- [2] One-way trip	0.7	OK

Visit the home page of BuildWise server, a red lava lamp is shown next to the project.



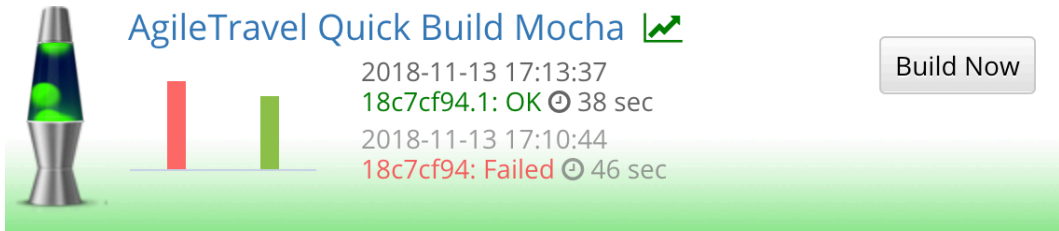
AgileTravel Quick Build Mocha

2018-11-13 17:10:44
18c7cf94: Failed 46 sec

Build Now

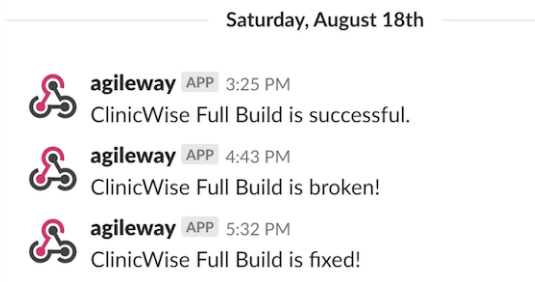
To fix a broken build, analyse the cause, update test scripts, and then trigger another build on BuildWise.

If all tests pass, you will be rewarded with a green lava lamp on the dashboard page.



Notification

You can configure continuous integration servers to send notifications such Emails and Slack immediately after a build has finished. The below is a set of Slack notifications sent from BuildWise:



Review

It is not that hard, is it? If you have been developing Selenium WebDriver tests using RSpec syntax, getting your tests running in BuildWise is quite straight forward. You can apply BuildWise configuration and the build script (**Rakefile**) of this sample project to your project with little modification.

Continuous Testing is becoming a hot topic in software industry, as it is the key process of DevOps. Obviously, this chapter is just an introduction, the main purpose is to show a simple and effective way of running your whole automated test suite, separately from your testing IDE. For more on this topic, please look out to my upcoming book: [Continuous Web and API Testing](https://leanpub.com/continuous-web-and-api-testing)⁵.

⁵<https://leanpub.com/continuous-web-and-api-testing>

Afterword

First of all, if you haven't downloaded the recipe test scripts from the book site, I strongly recommend you to do so. It is free for readers who have purchased the ebook through Leanpub.

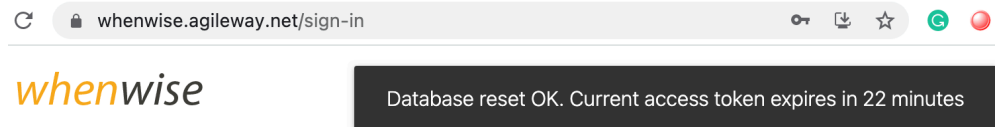
This book comes with two formats: *Ebook* and *Paper book*. I originally thought there won't be much demand for printed book, as the convenient 'search ability' of ebooks is good for this kind of solution books. However, during on-site consultation, I found some testers I worked with kept borrowing my printed proof-copy and wanted to buy it. It's why I released the paper book on Amazon as well.

Practice makes perfect

- Write tests

Many testers would like to practise test automation with Selenium WebDriver, but they don't have a good target application to write tests against. Here I make one of my applications available for you: [WhenWise sandbox site](https://whenwise.agileway.net)⁶. WhenWise is a modern web application using popular web technologies such as AJAX and Material Design. I have written 500 Selenium WebDriver tests for WhenWise. Execution of all tests takes more than 5 hours on a single machine. If you like, you can certainly practise writing tests against WhenWise sandbox server.

WhenWise is also a show case of web applications designed (based on the popular Material Design) for testing, which means it is easier to write automated tests against it. Our every Selenium test starts with calling a database reset: visit <https://whenwise.agileway.net/reset>, which will reset the database to a seeded state.



⁶<https://whenwise.agileway.net>

- **Improve programming skills**

It requires programming skills to effectively use Selenium WebDriver. For readers with no programming background, the good news is that the programming knowledge required for writing test scripts is much less comparing to coding applications, as you have seen in this book. If you like learning with hands-on practices, check out [Learn Ruby Programming by Examples](https://leanpub.com/learn-ruby-programming-by-examples-en)⁷.

Successful Test Automation

I believe that you are well equipped to cope with most testing scenarios if you have mastered the recipes in this book. However, this only applies to your ability to write individual tests. Successful test automation also requires developing and maintaining many automated test cases while software applications change frequently.

- **Maintain test scripts to keep up with application changes**

Let's say you have 100 automated tests that all pass. The changes developers made in the next build will affect some of your tests. As this happens too often, many automated tests will fail. The only way to keep the test script maintainable is to adopt good test design practices (such as reusable functions and page objects) and efficient refactoring. Check out my other book [Practical Web Test Automation](https://leanpub.com/practical-web-test-automation)⁸.

- **Shorten test execution time to get quick feedback**

With growing number of test cases, so is the test execution time. This leads to a long feedback gap from the time programmers committed the code to the time test execution completes. If programmers continue to develop new features/fixes during the gap time, it can easily get into a tail-chasing problem. This will hurt the team's productivity badly. Executing automated tests in a Continuous Testing server with various techniques (such as distributing test to run in parallel) can greatly shorten the feedback time. Check out my other book [Practical Continuous Testing](https://leanpub.com/practical-continuous-testing)⁹.

Best wishes for your test automation!

⁷<https://leanpub.com/learn-ruby-programming-by-examples-en>

⁸<https://leanpub.com/practical-web-test-automation>

⁹<https://leanpub.com/practical-continuous-testing>

Resources

Recipe test scripts

<http://zhimin.com/books/bought-selenium-recipes-nodejs>¹⁰

Username: agileway

Password: LOADWISE10

Log in with the above, or scan QR Code to access directly.



Books

- **Practical Web Test Automation**¹¹ by Zhimin Zhan

Solving individual selenium challenges (what this book is for) is far from achieving test automation success. *Practical Web Test Automation* is the book to guide you to the test automation success, topics include:

- Page object model
- Functional Testing Refactorings
- Cross-browser testing against IE, Firefox and Chrome
- Strategies on team collaboration and test automation adoption in projects and organizations

- **Practical Continuous Testing**¹² by Zhimin Zhan

Continuous Testing (CT) is the key process of DevOps. This books guides you to manage executing your Selenium tests in a CT server.

- **Selenium WebDriver Recipes in Java**¹³ by Zhimin Zhan

Sometimes you might be required to write Selenium WebDriver tests in Java. Master Selenium WebDriver in Java quickly by leveraging this book.

¹⁰<http://zhimin.com/books/bought-selenium-recipes-nodejs>

¹¹<https://leanpub.com/practical-web-test-automation>

¹²<https://leanpub.com/practical-continuous-testing>

¹³<https://leanpub.com/selenium-recipes-in-java>

- **Selenium WebDriver Recipes in C#, 2nd Edition**¹⁴ by Zhimin Zhan

Selenium WebDriver recipe tests in C#, another popular language that is quite similar to Java.

- **Selenium WebDriver Recipes in Ruby**¹⁵ by Zhimin Zhan

Selenium WebDriver tests can also be written in Ruby, a beautiful dynamic language very suitable for scripting tests. Master Selenium WebDriver in Ruby quickly by leveraging this book.

- **Selenium WebDriver Recipes in Python**¹⁶ by Zhimin Zhan

Selenium WebDriver recipes in Python, a popular script language that is similar to Ruby.

- **API Testing Recipes in Ruby**¹⁷ by Zhimin Zhan

The problem solving guide to testing APIs such as SOAP and REST web services in Ruby language.

Web Sites

- Selenium JavaScript API <https://seleniumhq.github.io/selenium/docs/api/javascript>¹⁸
- Selenium JavaScript Github¹⁹
- Selenium Home²⁰

Blog

- TestWisely Blog (<https://zhiminzhan.medium.com>)²¹

I share my experience and views on Test Automation and Continuous Testing there.

¹⁴<http://www.apress.com/9781484217412>

¹⁵<https://leanpub.com/selenium-recipes-in-ruby>

¹⁶<https://leanpub.com/selenium-recipes-in-python>

¹⁷<https://leanpub.com/api-testing-recipes-in-ruby>

¹⁸<https://seleniumhq.github.io/selenium/docs/api/javascript>

¹⁹<https://github.com/SeleniumHQ/selenium/tree/master/javascript/node/selenium-webdriver>

²⁰<https://www.selenium.dev>

²¹<https://zhiminzhan.medium.com>

Tools

- **Visual Studio Code**²²

Free and flexible code editor from Microsoft.

- **WebStorm IDE**²³

Commerical JavaScript IDE from JetBrains.

- **BuildWise** (<http://testwisely.com/buildwise>)²⁴

AgileWay's free and open-source continuous testing server, purposely designed for running automated UI tests with quick feedback.

²²<https://code.visualstudio.com/>

²³<https://www.jetbrains.com/webstorm/>

²⁴<http://testwisely.com/buildwise>