Selenium WebDriver Recipes in Java

The Problem Solving Guide to Selenium WebDriver

Zhimin Zhan

Selenium WebDriver Recipes in Java

The problem solving guide to Selenium WebDriver in Java

Zhimin Zhan

This book is for sale at http://leanpub.com/selenium-recipes-in-java

This version was published on 2016-02-02

To Dominic and Courtney!

Contents

Preface

After observing many failed test automation attempts by using expensive commercial test automation tools, I am delighted to see that the value of open-source testing frameworks has finally been recognized. I still remember the day (a rainy day at a Gold Coast hotel in 2011) when I found out that the Selenium WebDriver was the most wanted testing skill in terms of the number of job ads on the Australia's top job-seeking site.

Now Selenium WebDriver is big in the testing world. We all know software giants such as Facebook and LinkedIn use it, immensely-comprehensive automated UI testing enables them pushing out releases several times a day[1]. However, from my observation, many software projects, while using Selenium WebDriver, are not getting much value from test automation, and certainly nowhere near its potential. A clear sign of this is that the regression testing is not conducted on a daily basis (if test automation is done well, it will happen naturally).

Among the factors contributing to test automation failures, a key one is that automation testers lack sufficient knowledge in the test framework. It is quite common to see some testers or developers get excited when they first create a few simple test cases and see them run in a browser. However, it doesn't take long for them to encounter some obstacles: such as being unable to automate certain operations. If one step cannot be automated, the whole test case does not work, which is the nature of test automation. Searching solutions online is not always successful, and posting questions on forums and waiting can be frustrating (usually, very few people seek professional help from test automation coaches). Not surprisingly, many projects eventually gave up test automation or just used it for testing a handful of scenarios.

The motivation of this book is to help motivated testers work better with Selenium. The book contains over 100 recipes for web application tests with Selenium WebDriver. If you have read one of my other books: *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

[1]http://www.wired.com/business/2013/04/linkedin-software-revolution/
[2]https://leanpub.com/practical-web-test-automation

3. **Run** the test case
4. **See** test execution in your browser

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) Java coding 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. This book supports this style of reading. Since the recipes are arranged according to their levels of complexity, readers will also be able to work through the book from the front to back if they are looking to learn test automation with Selenium.

Recipe test scripts

To help readers to learn more effectively, this book has a dedicated site[3] that contains the recipe test scripts and related resources.

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, 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 2 (aka Selenium WebDriver) compliant, and can be run on Firefox, Chrome 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-java

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

January 2014

1. Introduction

Selenium is a free and open source library for automated testing web applications. I assume that you have had some knowledge of Selenium, based on the fact that you picked up this book (or opened it in your eBook reader).

Selenium

Selenium was originally created in 2004 by Jason Huggins, who was later joined by his other ThoughtWorks colleagues. Selenium supports all major browsers and tests can be written in many programming languages and run on Windows, Linux and Macintosh platforms.

Selenium 2 is merged with another test framework WebDriver (that's why you see 'selenium-webdriver') led by Simon Stewart at Google (update: Simon now works at FaceBook), Selenium 2.0 was released in July 2011.

Selenium language bindings

Selenium tests can be written in multiple programming languages such as Java, C#, Python and Ruby (the core ones). All examples in this book are written in Selenium with Java binding. As you will see 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 four different language bindings: Java, C#, Python and Ruby.

Java:

```
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) {
    // Create a new instance of the html unit driver
    // Notice that the remainder of the code relies on the interface,
    // not the implementation.
    WebDriver driver = new FirefoxDriver();

    // And now use this to visit Google
    driver.get("http://www.google.com");

    // Find the text input element by its name
    WebElement element = driver.findElement(By.name("q"));

    // Enter something to search for
    element.sendKeys("Hello Selenium WebDriver!");

    // Submit the form based on an element in the form
    element.submit();

    // Check the title of the page
    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
```

Set up Development Environment

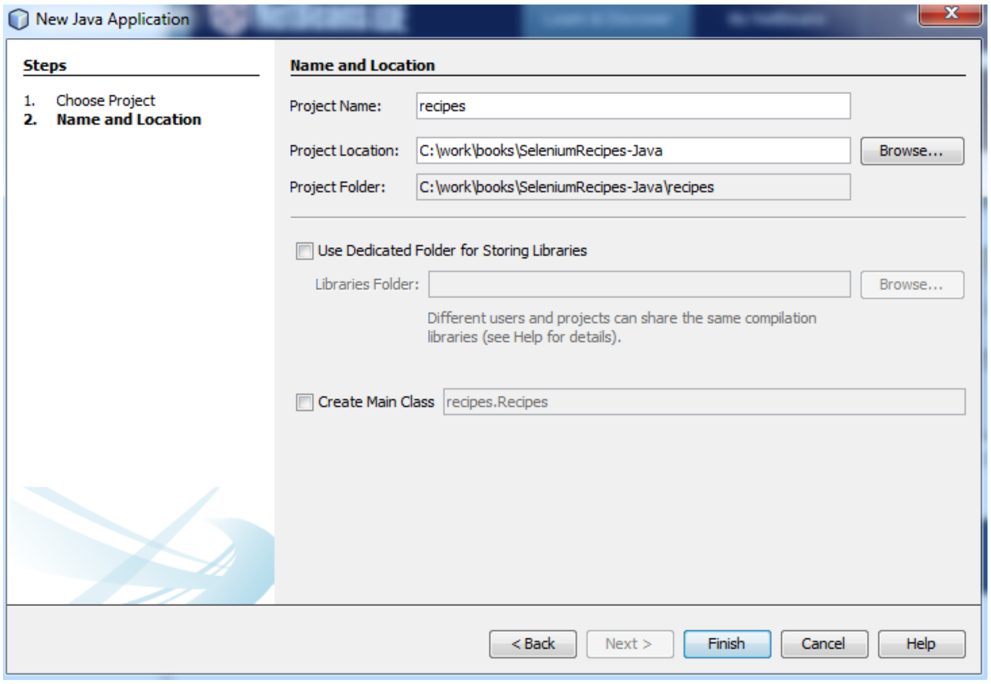
Most of Java programmers develop Java code in an IDE (integrated development environment), such as Eclipse and NetBeans. I will use NetBeans as the Java IDE of choice for this book, as all IDE related functions mentioned in the book are very generic, readers can easily apply in their favourite IDEs.

Prerequisite:

- Download and install JDK (jdk7 is the version used in recipes).
- Download and install NetBeans IDE.
- Download Selenium Java binding, eg. selenium-java-2.44.0.zip, about 24MB in size.
- Download and install Apache Ant, for running tests or test suites from command line.
- Your target browser is installed, such as Chrome or Firefox.

Set up NetBeans project

1. Create a new project in NetBeans


New Java Application

Steps
1. Choose Project
2. **Name and Location**

Name and Location

Project Name: recipes

Project Location: C:\work\books\SeleniumRecipes-Java

Project Folder: C:\work\books\SeleniumRecipes-Java\recipes

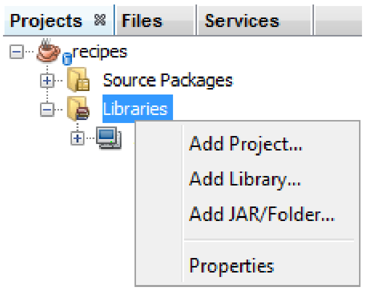
Use Dedicated Folder for Storing Libraries

Libraries Folder:

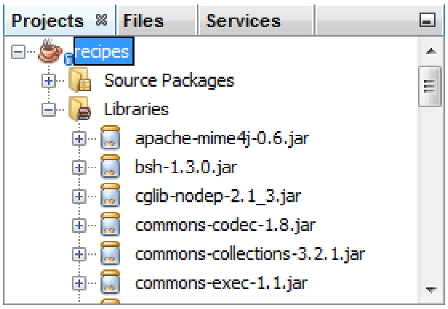
Different users and projects can share the same compilation libraries (see Help for details).

Create Main Class recipes.Recipes

2. Unzip selenium-java-VERSION.zip to copy all jar files (including ones under libs) to project's lib folder.

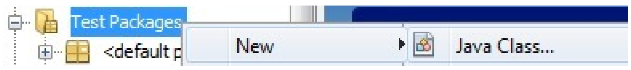


Here is what look like in NetBeans after all jar files added

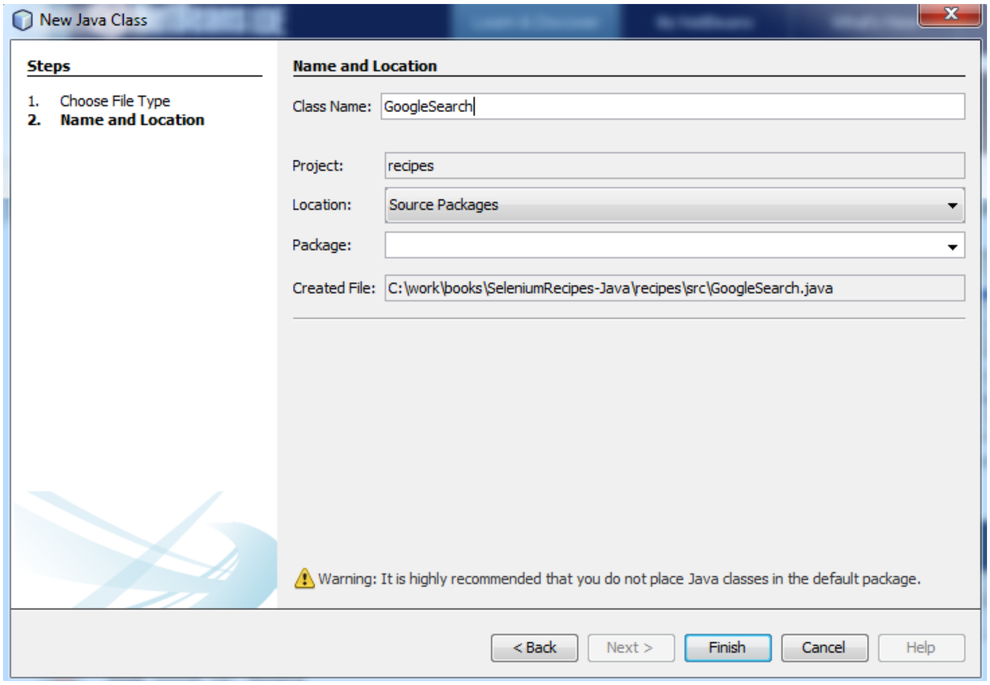


Create a test and run it

1. Add a new Java class (a test)
   Essentially a Selenium test in Java is a Java Class. Right click 'Test Packages' (not 'Source Packages', we are writing tests) to add a new Java class.



   Enter a name in Camel Case (such as SayHelloWorld)



Name and Location

Class Name: GoogleSearch

Project: recipes

Location: Source Packages

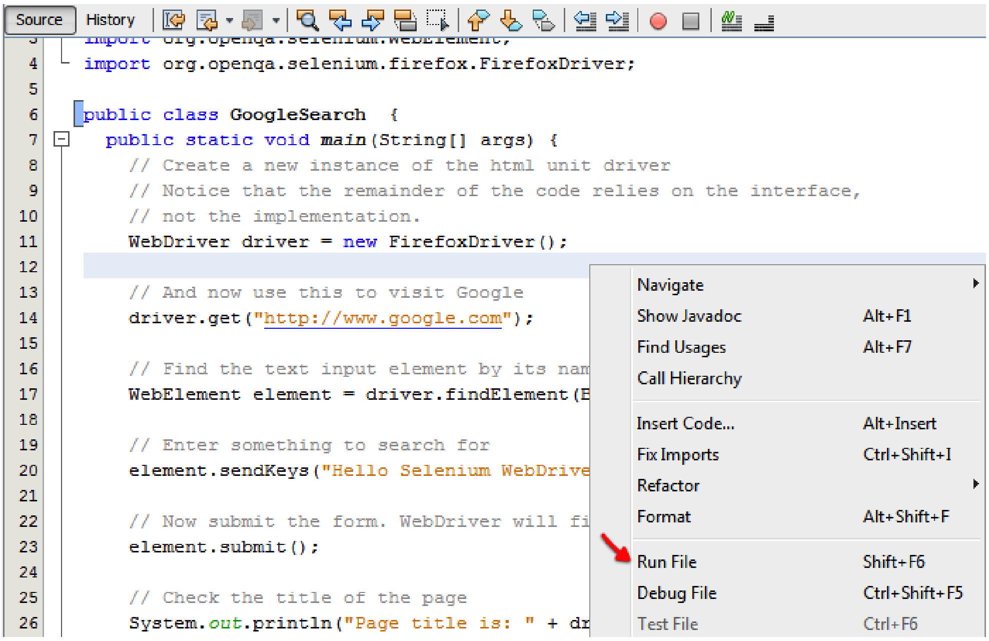
Package:

Created File: C:\work\books\SeleniumRecipes-Java\recipes\src\GoogleSearch.java

Warning: It is highly recommended that you do not place Java classes in the default package.

   In the example, we paste the Google Search test scripts in the editor.

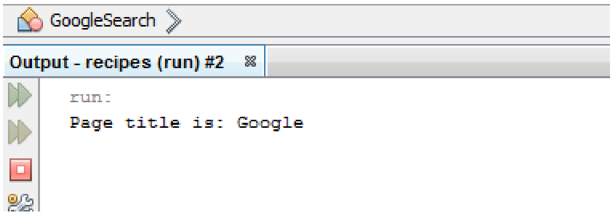
2. Right click the editor and select 'Run File'



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

public class GoogleSearch  {
  public static void main(String[] args) {
    // Create a new instance of the html unit driver
    // Notice that the remainder of the code relies on the interface,
    // not the implementation.
    WebDriver driver = new FirefoxDriver();

    // And now use this to visit Google
    driver.get("http://www.google.com");

    // Find the text input element by its nam
    WebElement element = driver.findElement(B

    // Enter something to search for
    element.sendKeys("Hello Selenium WebDrive

    // Now submit the form. WebDriver will fi
    element.submit();

    // Check the title of the page
    System.out.println("Page title is: " + dr
```

You shall see a Firefox browser is opening and do a 'google search' in it, as we our instruction (in the GoogleSearch.java).

The NetBeans console output will show the output generated from your Java class.



```
run:
Page title is: Google
```

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. The browser market nowadays is more diversified (based on the StatsCounter[1], the usage share in November 2014 for Chrome, IE and Firefox are 51.8%, 21.7% and 18.5% respectively). It is logical that all

[1]http://en.wikipedia.org/wiki/Usage_share_of_web_browsers

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.

Firefox

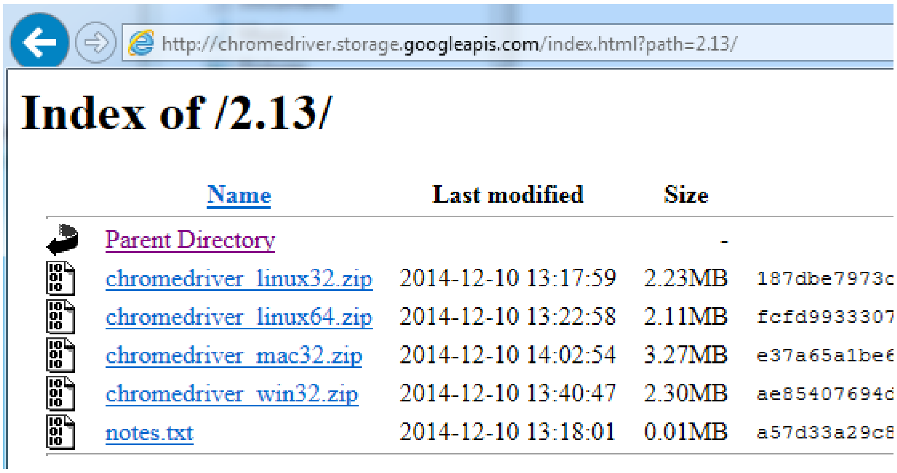
Selenium supports Firefox out of the box, i.e., as long as you have Firefox (and a not too outdated version) installed, you are ready to go. The test script below will open a web site in a new Firefox window.

```
import org.openqa.selenium.firefox.FirefoxDriver;
// ...
WebDriver driver = new FirefoxDriver();
```

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[2]



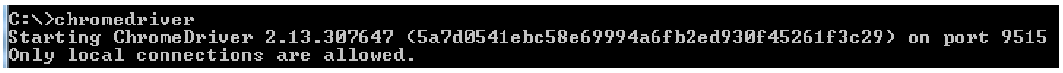
http://chromedriver.storage.googleapis.com/index.html?path=2.13/

Index of /2.13/

	Name	Last modified	Size	
	Parent Directory		-	
	chromedriver_linux32.zip	2014-12-10 13:17:59	2.23MB	187dbe7973c
	chromedriver_linux64.zip	2014-12-10 13:22:58	2.11MB	fcfd9933307
	chromedriver_mac32.zip	2014-12-10 14:02:54	3.27MB	e37a65a1be6
	chromedriver_win32.zip	2014-12-10 13:40:47	2.30MB	ae85407694d
	notes.txt	2014-12-10 13:18:01	0.01MB	a57d33a29c8

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:

[2]http://chromedriver.storage.googleapis.com/index.html



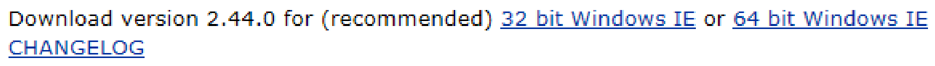
```
C:\>chromedriver
Starting ChromeDriver 2.13.307647 (5a7d0541ebc58e69994a6fb2ed930f45261f3c29) on port 9515
Only local connections are allowed.
```

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

```
import org.openqa.selenium.chrome.ChromeDriver;
//...
WebDriver driver = new ChromeDriver();
```

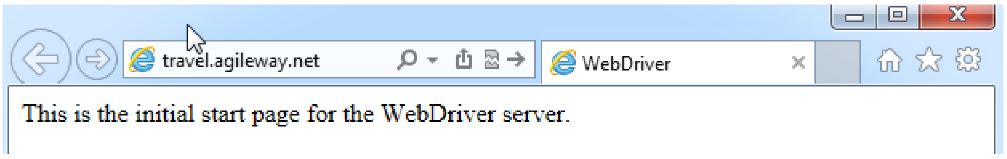
Internet Explorer

Selenium requires IEDriverServer to drive IE browser. Its installation process is very similar to *chromedriver*. IEDriverServer is available at http://www.seleniumhq.org/download/[3]. Choose the right one based on your windows version (32 or 64 bit).



Download version 2.44.0 for (recommended) 32 bit Windows IE or 64 bit Windows IE
CHANGELOG

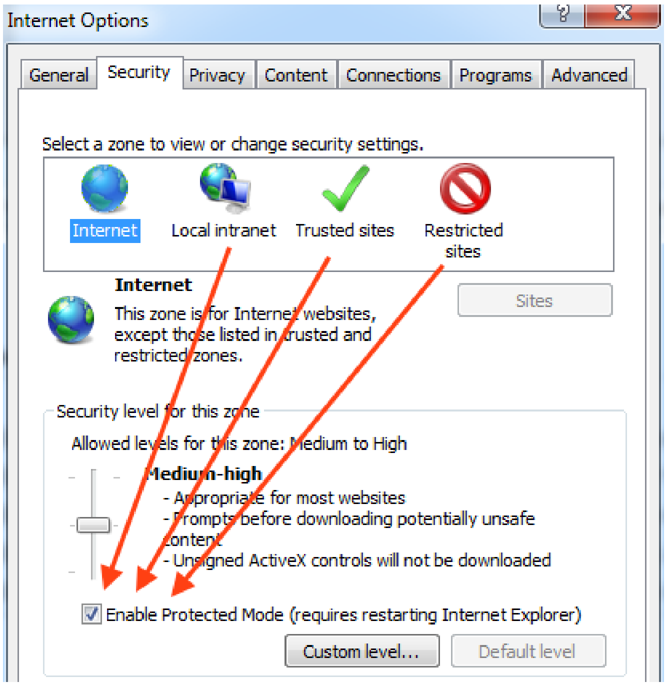
When a tests starts to execute in IE, before navigating the target test site, you will see this first:



This is the initial start page for the WebDriver server.

If you get this on IE9: "Unexpected error launching Internet Explorer. Protected Mode must be set to the same value (enabled or disabled) for all zones." Go to 'Internet Options', select each zone (as illustrated below) and make sure they are all set to the same mode (protected or not).

[3]http://www.seleniumhq.org/download/


Internet Options

General | Security | Privacy | Content | Connections | Programs | Advanced

Select a zone to view or change security settings.

Internet | Local intranet | Trusted sites | Restricted sites

Internet
This zone is for Internet websites, except those listed in trusted and restricted zones.

Sites

Security level for this zone
Allowed levels for this zone: Medium to High

Medium-high
- Appropriate for most websites
- Prompts before downloading potentially unsafe content
- Unsigned ActiveX controls will not be downloaded

☑ Enable Protected Mode (requires restarting Internet Explorer)

Custom level... | Default level

Further configuration is required for IE10 and IE11, see IE and IEDriverServer Runtime Configuration[4] for details.

```java
import org.openqa.selenium.ie.InternetExplorerDriver;
//...
WebDriver driver = new InternetExplorerDriver();
```

Edge

Edge is Mircosoft's new and default web browser on Windows 10. To drive Edge with WebDriver, you need download MicrosoftWebDriver server[5]. After installation, you will find the executable (*MicrosoftWebDriver.exe*) under *Program Files* folder, add it to your PATH.

[4]https://code.google.com/p/selenium/wiki/InternetExplorerDriver#Required_Configuration
[5]https://www.microsoft.com/en-us/download/details.aspx?id=48212

```java
import org.openqa.selenium.edge.EdgeDriver;
//...
WebDriver driver = new EdgeDriver();
```

JUnit

The examples above 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 framework that defines test structures and provides assertions (performing checks in test scripts). The de facto test framework for Java is JUnit, and here is an example using JUnit 4.

```java
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.support.pagefactory.*;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.ie.InternetExplorerDriver;

/**
 * Start 4 difference browsers by Selenium
 */
public class GoogleSearchDifferentBrowsersTest {

    @Test
    public void testInIE() throws Exception {
        WebDriver driver = new InternetExplorerDriver();
        driver.get("http://testwisely.com/demo");
        Thread.sleep(1000);
        driver.quit();
    }

    @Test
    public void testInFirefox() throws Exception {
        WebDriver driver = new FirefoxDriver();
        driver.get("http://testwisely.com/demo");
        Thread.sleep(1000);
        driver.quit();
    }


    @Test
    public void testInChrome() throws Exception {
        WebDriver driver = new ChromeDriver();
        driver.get("http://testwisely.com/demo");
        Thread.sleep(1000);
        driver.quit();
    }

    @Test
    public void testInEdge() throws Exception {
        WebDriver driver = new EdgeDriver();
        driver.get("http://testwisely.com/demo");
        Thread.sleep(1000);
        driver.quit();
    }
}
```

`@Test` annotates a test case below, in a format of `testCamelCase()`. You will find more about JUnit from its home page[6]. However, I honestly don't think it is necessary. The part used for test scripts is not much and quite intuitive. After studying and trying out some examples, you will be quite comfortable with JUnit.

JUnit fixtures

If you worked with xUnit before, you must know `setUp()` and `tearDown()` fixtures, used run before or after every test. In JUnit 4, by using annotations (`@BeforeClass`, `@Before`, `@After`, `@AfterClass`), you can choose the name for fixtures. Here are mine:

[6]http://junit.org/

```
@BeforeClass
public static void beforeAll() throws Exception {
  // run before all test cases
}

@Before
public void before() throws Exception {
  // run before each test case
}


@Test
public void testCase1() throws Exception {
  // one test case
}

@Test
public void testCase2() throws Exception {
  // another test case
}

@After
public void after() throws Exception {
  // run after each test case
}

@AfterClass
public static void afterAll() throws Exception {
  // run after all test cases
}
```

Run recipe scripts

Test scripts for all recipes can be downloaded from the book site. They are all in 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. To run tests written for a live site requires Internet connection.

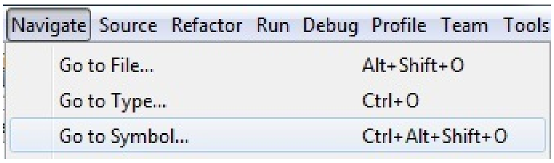
Run tests in NetBeans IDE

The most convenient way to run one test case or a test suite is to do it in an IDE. (When you have a large number of test cases, then the most effective way to run all tests is done by a Continuous Integration process)

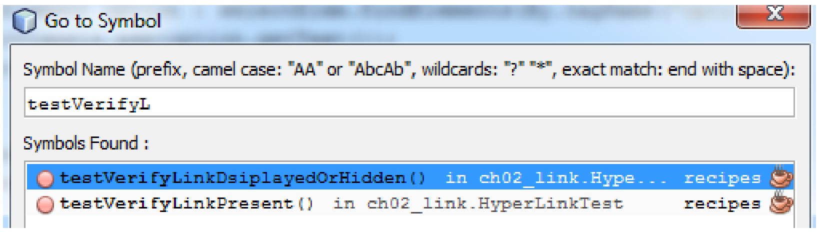
Find the test case

You can locate the recipe either by following the chapter or searching by name. There are over 100 test cases in one test project. Here is the quickest way to find the one you want in NetBeans.

Select menu 'Navigation' → 'Go to Symbol …'.



A pop up window lists all test cases in the project for your selection. The finding starts as soon as you type.

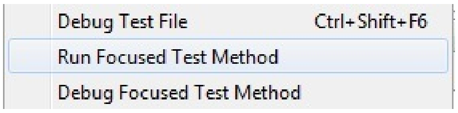

Go to Symbol
Symbol Name (prefix, camel case: "AA" or "AbcAb", wildcards: "?" "*", exact match: end with space):
Symbols Found :

```
testVerifyL
```

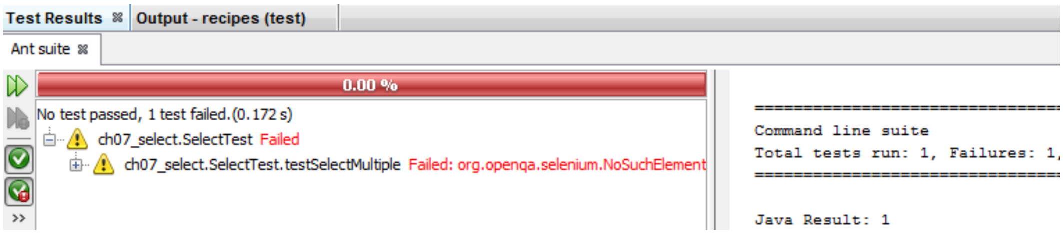
```
testVerifyLinkDsiplayedOrHidden()  in ch02_link.Hype...  recipes
testVerifyLinkPresent()  in ch02_link.HyperLinkTest      recipes
```

Run individual test case

Move caret to a line within a test case (between `public void testXXX() throws Exception {` and `}`). Right mouse click and select "Run Focused Test Method" to run this case.



The below is a screenshot of execution panel when one test case failed,

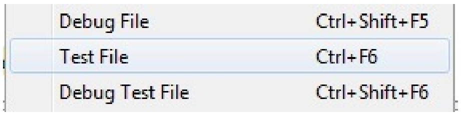


```
Test Results    Output - recipes (test)
Ant suite
0.00 %
No test passed, 1 test failed.(0.172 s)
ch07_select.SelectTest Failed
  ch07_select.SelectTest.testSelectMultiple Failed: org.openqa.selenium.NoSuchElement
```

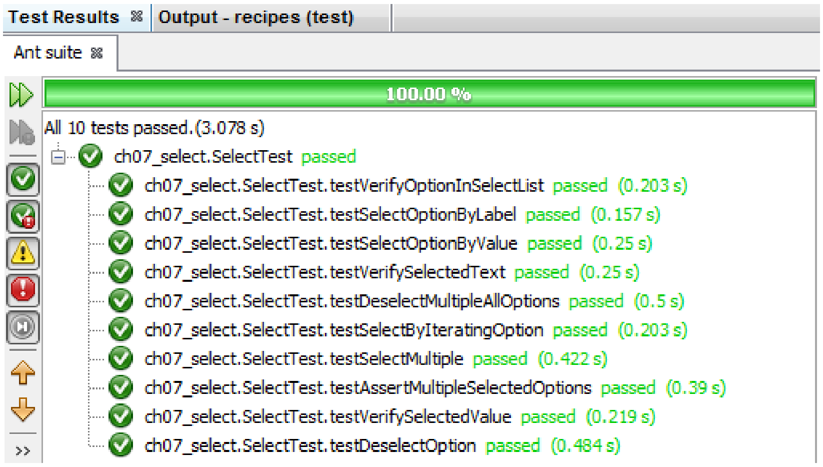
```
===================================
Command line suite
Total tests run: 1, Failures: 1,
===================================

Java Result: 1
```

Run all test cases in a test script file

You can also run all test cases in the currently opened test script file by right mouse clicking anywhere in the editor and selecting 'Test File'. (Ctrl+F6)



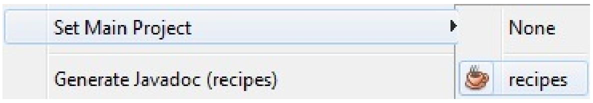
The below is a screenshot of the execution panel when all test cases in a test script file passed,



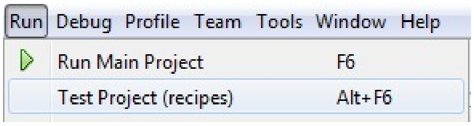
```
Test Results    Output - recipes (test)
Ant suite
100.00 %
All 10 tests passed.(3.078 s)
ch07_select.SelectTest passed
  ch07_select.SelectTest.testVerifyOptionInSelectList passed (0.203 s)
  ch07_select.SelectTest.testSelectOptionByLabel passed (0.157 s)
  ch07_select.SelectTest.testSelectOptionByValue passed (0.25 s)
  ch07_select.SelectTest.testVerifySelectedText passed (0.25 s)
  ch07_select.SelectTest.testDeselectMultipleAllOptions passed (0.5 s)
  ch07_select.SelectTest.testSelectByIteratingOption passed (0.203 s)
  ch07_select.SelectTest.testSelectMultiple passed (0.422 s)
  ch07_select.SelectTest.testAssertMultipleSelectedOptions passed (0.39 s)
  ch07_select.SelectTest.testVerifySelectedValue passed (0.219 s)
  ch07_select.SelectTest.testDeselectOption passed (0.484 s)
```

Run all tests

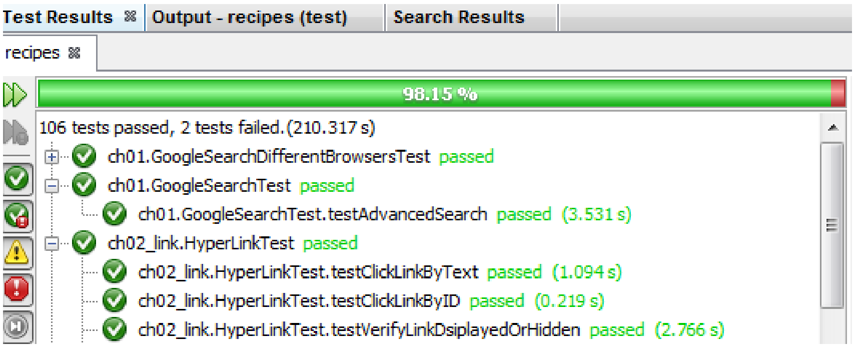
You can also run all test cases in a NetBeans project. Firstly, set the main project by selecting menu 'Run' → 'Set Main Project' → 'Your Project Name' (once off).



Then select 'Run' → 'Test Project' to trigger a run of all test cases in this project.



The below is a screenshot of the test results panel after running over 100 tests across dozens of test files.



106 tests passed, 2 tests failed.(210.317 s)
ch01.GoogleSearchDifferentBrowsersTest passed
ch01.GoogleSearchTest passed
ch01.GoogleSearchTest.testAdvancedSearch passed (3.531 s)
ch02_link.HyperLinkTest passed
ch02_link.HyperLinkTest.testClickLinkByText passed (1.094 s)
ch02_link.HyperLinkTest.testClickLinkByID passed (0.219 s)
ch02_link.HyperLinkTest.testVerifyLinkDsiplayedOrHidden passed (2.766 s)

Run tests from command line

One key advantage of open-source test frameworks, such as Selenium, is FREEDOM. You can edit the test scripts in any text editors and run them from a command line.

To run a Java class, you needs to compile it first (Within IDE, IDEs do it for you automatically). Running code in compiled language (such as Java) with many libraries dependency from command line is not easy as dynamic ones (such as Ruby). Build tools such as Ant can help on this.

I included an Ant build.xml (with recipe source) to simplify the test execution from command line. To run test cases in a test script file (named ch09_assertion.AssertionTest.java), enter command

```
> ant runTest -DTestName=ch09_assertion.AssertionTest
```

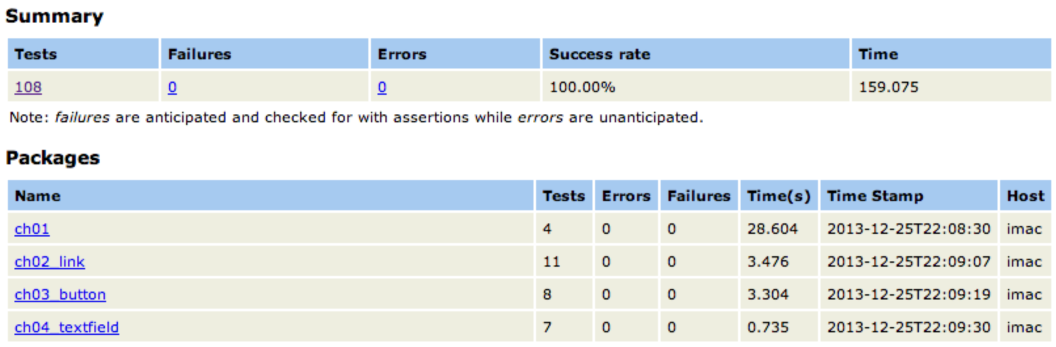
Example Output

```
compile:
    [mkdir] Created dir: /Users/zhimin/books/SeleniumRecipes-Java/recipes/bu\
ild/classes
    [javac] Compiling 22 source files to /Users/zhimin/books/SeleniumRecipes\
-Java/recipes/build/classes

runTest:
    [junit] Running ch09_assertion.AssertionTest
    [junit] Tests run: 11, Failures: 0, Errors: 0, Time elapsed: 0.659 sec

BUILD SUCCESSFUL
Total time: 9 seconds
```

Also, to run all recipe tests (within the test folder)

```
> ant runAll
```

which generate JUnit style test report like this



Summary

Tests	Failures	Errors	Success rate	Time
108	0	0	100.00%	159.075

Note: *failures* are anticipated and checked for with assertions while *errors* are unanticipated.

Packages

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
ch01	4	0	0	28.604	2013-12-25T22:08:30	imac
ch02_link	11	0	0	3.476	2013-12-25T22:09:07	imac
ch03_button	8	0	0	3.304	2013-12-25T22:09:19	imac
ch04_textfield	7	0	0	0.735	2013-12-25T22:09:30	imac

The command syntax is identical for Windows, Mac OS X and Linux platforms.

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 uses what is called locators to find and match the elements on web page. There are 8 locators in Selenium:

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

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

Start browser

Testing web sites starts with a browser.

```
static WebDriver driver = new FirefoxDriver();
driver.get("http://testwisely.com/demo")
```

Use `ChromeDriver` and `IEDriver` for testing in Chrome and IE respectively.

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

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

Find element by ID

Using IDs is the easiest and the safest way to locate an element in HTML. If the page is W3C HTML conformed[1], the IDs should be unique and identified in web controls. In comparison

[1]http://www.w3.org/TR/WCAG20-TECHS/H93.html

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).

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

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");
```

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();
```

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();
```

Find element by XPath

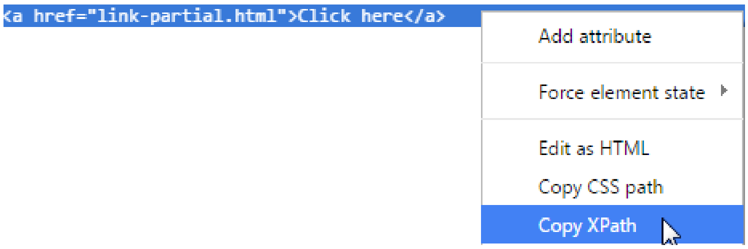
XPath, the XML Path Language, is a query language for selecting nodes from an XML document. When a browser renders a web page, it parses it into a DOM tree or similar. XPath can be used to refer a certain node in the DOM tree. If this sounds a little too much technical for you, don't worry, just remember 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']")).clic\
k();
```

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):


<a href="link-partial.html">Click here</a>

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.

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 `tag_name` 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.

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-deault 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 below will return error "Compound class names not permitted"
// driver.findElement((By.className("btn btn-deault btn-primary")).click();
```

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();
Thread.sleep(500);
driver.findElement(By.className("editable-textarea")).sendKeys("inline notes\
");
Thread.sleep(500);
driver.findElement(By.className("editable-submit")).click();
```

Find element by CSS Selector

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

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

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

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();
```

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.

```
List<WebElement> checkbox_elems = driver.findElements(By.xpath("//div[@id='c\
ontainer']//input[@type='checkbox']"));
System.out.println(checkbox_elems); // => 2
checkbox_elems.get(1).click();
```

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>
```

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.findElement(By.linkText("Recommend Selenium")).click();
```

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

Click a link by partial text

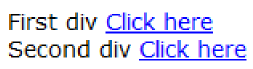
```
driver.findElement(By.partialLinkText("Recommend Seleni")).click();
```

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>
```

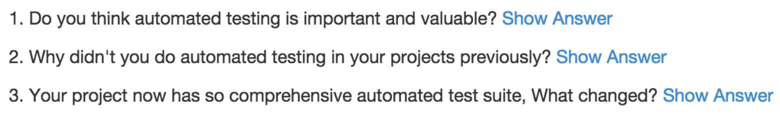
If test case requires you to click the second 'Click here' link, the simple `findElement(By.linkText("Click here"))` won't work (as it clicks the first one). Here is a way to accomplish using XPath:

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

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 three 'Show Answer" links,



1. Do you think automated testing is important and valuable? Show Answer
2. Why didn't you do automated testing in your projects previously? Show Answer
3. Your project now has so comprehensive automated test suite, What changed? Show Answer

To click the second one,

```
assert driver.findElements(By.linkText("Show Answer")).size() == 2;
driver.findElements(By.linkText("Show Answer")).get(1).click(); // 2nd link
```

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

Click Nth link by CSS Selector

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

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

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

Verify a link present or not?

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

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.

```
WebElement seleniumLink = driver.findElement(By.linkText("Recommend Selenium\
"));
assert seleniumLink.getAttribute("href").equals(TestHelper.siteUrl() + "inde\
x.html");
assert "recommend_selenium_link".equals(seleniumLink.getAttribute("id"));
assert "Recommend Selenium".equals(seleniumLink.getText());
assert "a".equals(seleniumLink.getTagName());
```

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

```
assert "font-size: 14px;".equals(driver.findElement(By.id("recommend_seleniu\
m_link")).getAttribute("style"));
//  Please note using attribute_value("style") won't work
assert "123".equals(driver.findElement(By.id("recommend_selenium_link")).get\
Attribute("data-id"));
```

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>
```

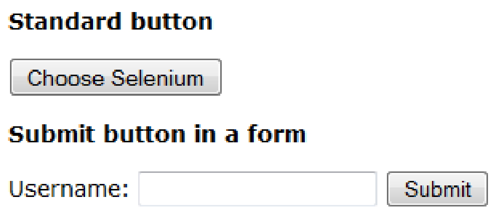
While we could use `switchTo()` method (see chapter 10) to find the new browser window, it will be easier to perform all testing within one browser window. Here is how:

```
String currentUrl = driver.getCurrentUrl();
String newWindowUrl = driver.findElement(By.linkText("Open new window")).get\
Attribute("href");
driver.navigate().to(newWindowUrl);
driver.findElement(By.name("name")).sendKeys("sometext");
driver.navigate().to(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).



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.

Click a button by label

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

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();
```

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.

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

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.

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();
```

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')]")).cli\
ck();
```

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.

```
WebElement a_btn = driver.findElement(By.id("choose_selenium_btn"));
((JavascriptExecutor) driver).executeScript("arguments[0].click();", a_btn);
```

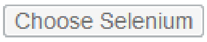
Assert a button present

Just like hyperlinks, we can use `displayed?` to check whether a control is present on a web page. This check applies to most of the web controls in Selenium.

```
assert driver.findElement(By.id("choose_selenium_btn")).isDisplayed();
driver.findElement(By.linkText("Hide")).click();
Thread.sleep(500);
assert !driver.findElement(By.id("choose_selenium_btn")).isDisplayed();
```

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.

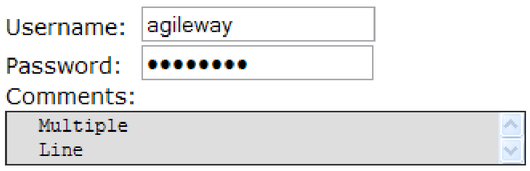


Normally enabling or disabling buttons (or other web controls) is triggered by JavaScripts.

```
assert driver.findElement(By.id("choose_selenium_btn")).isEnabled();
driver.findElement(By.linkText("Disable")).click();
Thread.sleep(500);
assert !driver.findElement(By.id("choose_selenium_btn")).isEnabled();
driver.findElement(By.linkText("Enable")).click();
Thread.sleep(500);
assert driver.findElement(By.id("choose_selenium_btn")).isEnabled();
```

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.



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>
```

Enter text into a text field by name

```
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.

Enter text into a text field by ID

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

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");
```

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");
```

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.

Assert value

```
driver.findElement(By.id("user")).sendKeys("testwisely");
assert "testwisely".equals(driver.findElement(By.id("user")).getAttribute("v\
alue"));
```

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.

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

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"/>
```

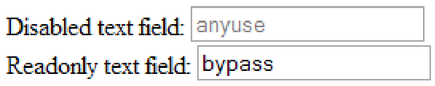
If a text box is set to be read-only, the following test step will not work.

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

Here is a workaround:

```
((JavascriptExecutor) driver).executeScript("$('#readonly_text').val('bypass\
');");
assert "bypass".equals(driver.findElement(By.id("readonly_text")).getAttribu\
te("value"));
((JavascriptExecutor) driver).executeScript("$('#disabled_text').val('anyuse\
');");
```

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: anyuse
Readonly text field: bypass

Set and assert the value of a hidden field

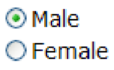
A hidden field is often used to store a default value.

```
<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.

```
WebElement theHiddenElem = driver.findElement(By.name("currency"));
assert theHiddenElem.getAttribute("value").equals("USD");
((JavascriptExecutor) driver).executeScript("arguments[0].value = 'AUD';", t\
heHiddenElem);
assert theHiddenElem.getAttribute("value").equals("AUD");
```

6. Radio button



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
```

Select a radio button

```
driver.findElement(By.xpath("//input[@name='gender' and @value='female']")).\
click();
Thread.sleep(500);
driver.findElement(By.xpath("//input[@name='gender' and @value='male']")).cl\
ick();
```

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();
```

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, another 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']")).\
click();
try {
    driver.findElement(By.xpath("//input[@name='gender' and @value='female']\
")).clear();
} catch (Exception ex) {
    // Selenium does not allow
    System.out.println("Selenium does not allow clear currently selected rad\
io button, just select another one");
    driver.findElement(By.xpath("//input[@name='gender' and @value='male']")\
).click();
}
```

Assert a radio option is selected

```
driver.findElement(By.xpath("//input[@name='gender' and @value='female']")).\
click();
assert driver.findElement(By.xpath("//input[@name='gender' and @value='femal\
e']")).isSelected();
assert !driver.findElement(By.xpath("//input[@name='gender' and @value='male\
']")).isSelected();
```

Iterate radio buttons in a radio group

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

```
assert driver.findElements(By.name("gender")).size() == 2;
for (WebElement rb : driver.findElements(By.name("gender"))) {
    if (rb.getAttribute("value").equals("female")) {
        rb.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.

Click Nth radio button in a group

```
driver.findElements(By.name("gender")).get(1).click();
assert driver.findElement(By.xpath("//input[@name='gender' and @value='femal\
e']")).isSelected();
driver.findElements(By.name("gender")).get(0).click();
assert driver.findElement(By.xpath("//input[@name='gender' and @value='male'\
]")).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.

Click radio button by the following label

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

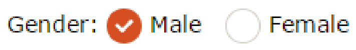
```
<div id="q1" class="question">
  <div class="question-answer col-lg-5">
    <div class="yes-no">
      <input id="QuestionViewModels_1__SelectedAnswerId" name="QuestionViewM\
odels[1].SelectedAnswerId" type="radio" value="c225306e-8d8e-45b0-8261-22617\
d9796b5">
      <label for="QuestionViewModels_1__SelectedAnswerId">Yes</label>
    </div>
    <div class="yes-no">
      <input id="QuestionViewModels_1__SelectedAnswerId" name="QuestionViewM\
odels[1].SelectedAnswerId" type="radio" value="85ff8db7-1c58-47a2-a978-58120\
0fb7098">
      <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.

```
WebElement elem = driver.findElement(By.xpath("//div[@id='q1']//label[contai\
ns(.,'Yes')]/../input[@type='radio']"));
elem.click();
```

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).



The iCheck JavaScript transforms the radio button HTML fragment

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

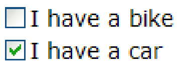
to

```
<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: Element is not clickable
// driver.findElement(By.id("q2_1")).click();
driver.findElements(By.className("iradio_square-red")).get(0).click();
driver.findElements(By.className("iradio_square-red")).get(1).click();

// More precise with XPath
driver.findElement(By.xpath("//div[contains(@class, 'iradio_square-red')]/in\
put[@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 h\
ave a bike<br>
<input type="checkbox" name="vehicle_car" id="checkbox_car">I have a car
```

Check by name

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

Check by id

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

Uncheck a checkbox

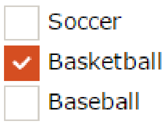
```
WebElement the_checkbox = driver.findElement(By.id("checkbox_car"));
the_checkbox.click();
if (the_checkbox.isSelected()) {
    the_checkbox.click();
}
```

Assert a checkbox is checked (or not)

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

Customized Checkboxes - iCheck

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



The iCheck JavaScript transforms the checkbox HTML fragment

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

to

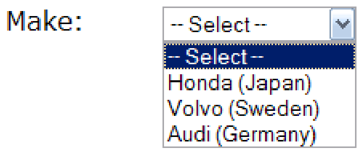
```
<div class="icheckbox_square-red" style="position: relative;">
   <input type="checkbox" name="sports[]" value="Soccer" style="..."/>
   <ins class="iCheck-helper" style="..."/>
</div>
```

Here are test scripts to drive iCheck checkboxes.

```
driver.findElements(By.className("icheckbox_square-red")).get(0).click();
driver.findElements(By.className("icheckbox_square-red")).get(1).click();

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

8. Select List

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



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>
</select>
```

Select an option by text

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

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

Select an option by value

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

```
Select select = new Select(driver.findElement(By.id("car_make_select")));
select.selectByValue("audi");
```

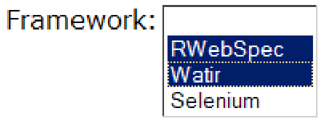
Select an option by iterating all options

Here I will show you a far 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.

```
WebElement selectElem = driver.findElement(By.id("car_make_select"));
for (WebElement option : selectElem.findElements(By.tagName("option"))) {
    if (option.getText().equals("Volvo (Sweden)")) {
        option.click();
    }
}
```

Select multiple options

A select list also supports multiple selections.



HTML Source

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

Test Script

```
Select select = new Select(driver.findElement(By.name("test_framework")));
select.selectByVisibleText("Selenium");
select.selectByValue("rwebspec");
select.selectByIndex(2);
assert select.getAllSelectedOptions().size() == 3;
```

Clear one selection

```
Select select = new Select(driver.findElement(By.name("test_framework")));
select.selectByVisibleText("RWebSpec");
select.selectByVisibleText("Selenium");
select.deselectByVisibleText("RWebSpec");
select.deselectByValue("selenium");
// one more
// select.deselectByIndex(0);
assert select.getAllSelectedOptions().size() == 0;
```

Clear selection

Clear selection works the same way for both single and multiple select lists.

```
Select select = new Select(driver.findElement(By.name("test_framework")));
select.selectByVisibleText("Selenium");
select.selectByVisibleText("RWebSpec");
select.deselectAll();
assert select.getAllSelectedOptions().size() == 0;
```

Assert selected option

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

```
Select select = new Select(driver.findElement(By.id("car_make_select")));
select.selectByValue("audi");
assert "Audi (Germany)".equals(select.getFirstSelectedOption().getText());
```

Assert the value of a select list

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

```
Select select = new Select(driver.findElement(By.id("car_make_select")));
select.selectByVisibleText("Volvo (Sweden)");
assert "volvo".equals(select.getFirstSelectedOption().getAttribute("value"));
```

Assert multiple selections

A multiple select list can have multiple options being selected.

```
Select select = new Select(driver.findElement(By.name("test_framework")));
select.selectByVisibleText("Selenium");
select.selectByVisibleText("RWebSpec");

List<WebElement> selected = select.getAllSelectedOptions();
assert selected.size() == 2;
assert "RWebSpec".equals(selected.get(0).getText()); // display order
assert "Selenium".equals(selected.get(1).getText());
```

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.

Go to a URL

```
driver.get("http://testwisely.com");
driver.navigate().to("https://google.com");
```

Visit pages within a site

`driver.navigate().to()` 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.

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

// ...

public void visit(String path) {
    driver.navigate().to(site_root_url + path);
}

@Test
public void testGoToPageWithinSiteUsingFunction() {
    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`.

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

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();
```

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(new Dimension(1024, 768));
```

Maximize browser window

```
driver.manage().window().maximize();
Thread.sleep(1000);  // wait 1 second to see the effect
driver.manage().window().setSize(new Dimension(1024, 768));
```

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(new Point(100, 100));
Thread.sleep(1000);
driver.manage().window().setPosition(new Point(0, 0));
```

Minimize browser window

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

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

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

Scroll focus to control

For certain controls are not viewable in a web page (due to JavaScript), WebDriver unables 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.

```
WebElement elem = driver.findElement(By.name("submit_action_2"));
Integer elemPos = elem.getLocation().getY();
((JavascriptExecutor) driver).executeScript("window.scroll(0, " + elemPos + \
");");
Thread.sleep(500);
elem.click();
```

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

```
driver.findElement(By.linkText("Open new window")).click();
Set<String> windowHandles = driver.getWindowHandles();
String firstTab = (String) windowHandles.iterator().next();
String lastTab = null;
for (Iterator iter = windowHandles.iterator(); iter.hasNext();) {
    lastTab = (String) iter.next();
}
driver.switchTo().window(lastTab);
assert driver.findElement(By.tagName("body")).getText().contains("This is ur\
l link page");
driver.switchTo().window(firstTab); // back to first tab/window
assert driver.findElement(By.linkText("Open new window")).isDisplayed();
```

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).

```
String url; // instance variable

@Before
public void before() throws Exception {
   url = driver.getCurrentUrl();
}

//...

@Test
public void testSelectOptionByLabel() throws Exception {
   driver.findElement(By.linkText("Button")).click();
   //...
   driver.navigate().to(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 `public void testXXX() {` to `}`).

In this example, the `url` variable is used in `@Before` 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:

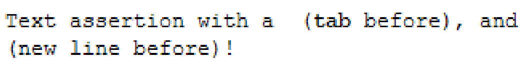
- page title (equals)
- page text (contains or does not contain)
- page source (contains or does not contain)
- input element value (equals)
- display element text (equals)
- element state (selected, disabled, displayed)

Assert page title

```
assert "TestWise IDE".equals(driver.getTitle());
```

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>
```

Test script

```
String matching_str = "Text assertion with a  (tab before), and \n(new line \
before)!";
assert driver.findElement(By.tagName("body")).getText().contains(matching_st\
r);
```

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.

Assert Page Source

The page source is raw HTML returned from the server.

```
String html_fragment = "Text assertion with a  (<b>tab</b> before), and \n(n\
ew line before)!";
assert driver.getPageSource().contains(html_fragment);
```

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().equals("First Label");
```

Assert Span text

HTML source

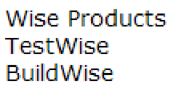
```
<span id="span_2">Second Span</span>
```

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

```
driver.findElement(By.id("span_2")).getText().equals("Second Span");
```

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>
```

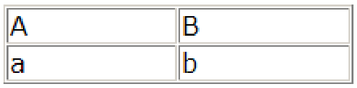
Test script

```
driver.findElement(By.id("div_child_1")).getText().equals("TestWise");
driver.findElement(By.id("div_parent")).getText().equals("Wise Products\nTes\
tWise\nBuildWise");
```

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

```
WebElement the_element = driver.findElement(By.id("alpha_table"));
assert "A B\na b".equals(the_element.getText());
Object html = ((JavascriptExecutor) driver).executeScript("return  arguments\
[0].outerHTML;", the_element);
assert ((String) html).contains("<td id=\"cell_1_1\">A</td>");
```

Assert text in a table cell

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

```
assert "A".equals(driver.findElement(By.id("cell_1_1")).getText());
```

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

```
assert "b".equals(driver.findElement(By.xpath("//table/tbody/tr[2]/td[2]")).\
getText());
```

Assert text in a table row

```
assert "A B".equals(driver.findElement(By.id("row_1")).getText());
```

Assert image present

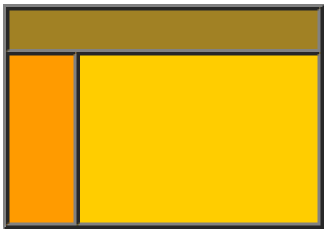
```
assert driver.findElement(By.id("next_go")).isDisplayed();
```

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.

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" ma\
rginwidth="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).

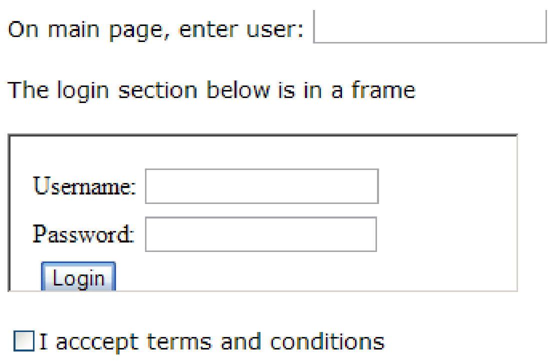
```
driver.switchTo().frame("topNav"); // name
driver.findElement(By.linkText("Menu 2 in top frame")).click();

// need to switch to default before another switch
driver.switchTo().defaultContent();
driver.switchTo().frame("menu_frame"); //fail on Chrome, fine for Firefox
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.

Testing iframe

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

Example page


On main page, enter user:

The login section below is in a frame

Username:

Password:

Login

I acccept terms and conditions

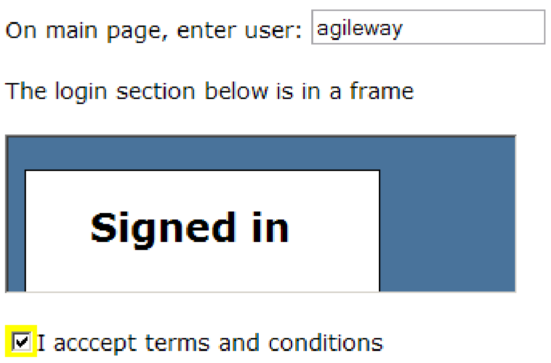
HTML Source

```
<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.switchTo().frame("Frame1"); // name
driver.findElement(By.name("username")).sendKeys("tester");
driver.findElement(By.name("password")).sendKeys("TestWise");
driver.findElement(By.id("loginBtn")).click();
assert driver.getPageSource().contains("Signed in");
driver.switchTo().defaultContent();
driver.findElement(By.id("accept_terms")).click();
```

The web page after test execution looks as below:


On main page, enter user: agileway

The login section below is in a frame

Signed in

[✓] I acccept terms and conditions

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

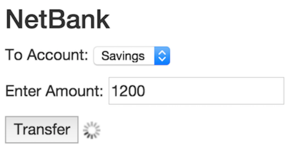
Test multiple iframes

A web page may contain multiple iframes.

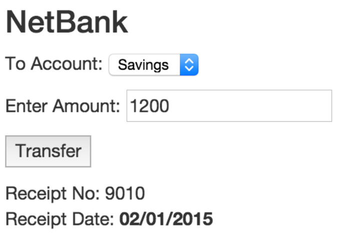
```
driver.switchTo().frame(0);
driver.findElement(By.name("username")).sendKeys("agileway");
driver.switchTo().defaultContent();
driver.switchTo().frame(1);
driver.findElement(By.id("radio_male")).click();
```

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:



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



Receipt No: 9010
Receipt Date: **02/01/2015**

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.

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.

```
driver.findElement(By.xpath("//input[@value='Transfer']")).click();
Thread.sleep(10000);
assert driver.findElement(By.tagName("body")).getText().contains("Receipt No\
:");
```

`Thread.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'.

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.

```
driver.findElement(By.xpath("//input[@value='Transfer']")).click();
WebDriverWait wait = new WebDriverWait(driver, 10);  // seconds
wait.until(ExpectedConditions.presenceOfElementLocated(By.id("receiptNo")));
```

Besides `presenceOfElementLocated`, there are many others such as `elementToBeClickable`, `presenceOfAllElementsLocatedBy`,..., etc. The full list can be found on this JavaDoc page[1].

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.

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

```
driver.findElement(By.xpath("//input[@value='Transfer']")).click();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
assert driver.findElement(By.id("receiptNo")).getText().length() > 0;
// reset for later steps
driver.manage().timeouts().implicitlyWait(0, TimeUnit.SECONDS);
```

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.

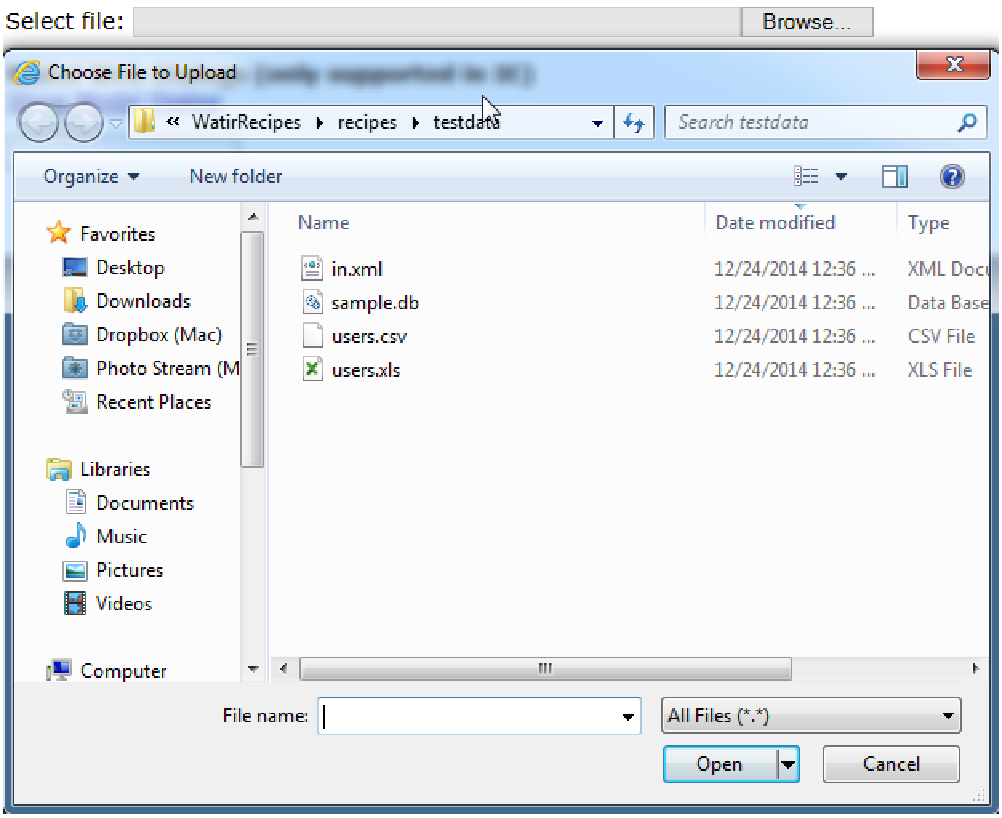
```
@Test
public void TestWaitUsingJQueryActiveFlag()  throws Exception {
  // ...
  driver.findElement(By.xpath("//input[@value='Pay now']")).click();
  waitForAjaxComplete(11);
  assert(driver.findElement(By.tagName("body")).getText().contains("Booking \
number"));
}

public void waitForAjaxComplete(int maxSeconds) throws Exception {
  boolean is_ajax_comlete = false;
  for (int i = 1; i <= maxSeconds; i++) {
    is_ajax_comlete = (boolean) ((JavascriptExecutor) driver).executeScript(\
"return jQuery.active == 0");
    if (is_ajax_comlete) {
      return;
    }
    Thread.sleep(1000);
  }
  throw new RuntimeException("Timed out after " + maxSeconds + " seconds");
}
```

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.

File upload

Example page



Name	Date modified	Type
in.xml	12/24/2014 12:36 ...	XML Docu
sample.db	12/24/2014 12:36 ...	Data Base
users.csv	12/24/2014 12:36 ...	CSV File
users.xls	12/24/2014 12:36 ...	XLS File

HTML Source

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

Test script

```
String filePath = "C:\\testdata\\logo.png";
driver.findElement(By.name("document[file]")).sendKeys(filePath);
```

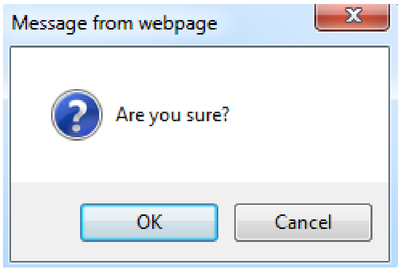
The first slash of \\ (for Windows platform) is for escaping the later one, the whole purpose is to pass the value "C:\testdata\logo.png" to the control.

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:

```
// scriptDir() is a helper function returns the current working folder
String filePath = TestHelper.scriptDir() +  File.separator + "testdata" +  F\
ile.separator + "users.csv";
driver.findElement(By.name("document[file]")).sendKeys(filePath);
```

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.findElement(By.xpath("//input[contains(@value, 'Buy Now')]")).click();
Alert a = driver.switchTo().alert();
if (a.getText().equals("Are you sure")) {
    a.accept();
} else {
    a.dismiss();
}
```

Handle JavaScript pop ups with JavaScript

```
((JavascriptExecutor) driver).executeScript("window.confirm = function() { r\
eturn true; }");
((JavascriptExecutor) driver).executeScript("window.alert = function() { ret\
urn true; }");
((JavascriptExecutor) driver).executeScript("window.prompt = function() { re\
turn 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]

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 (JUnit 4 feature) with a specified maximum time.

```
@Test(timeout = 5000)
public void testTimeout() throws Exception {
  // operations here
}
```

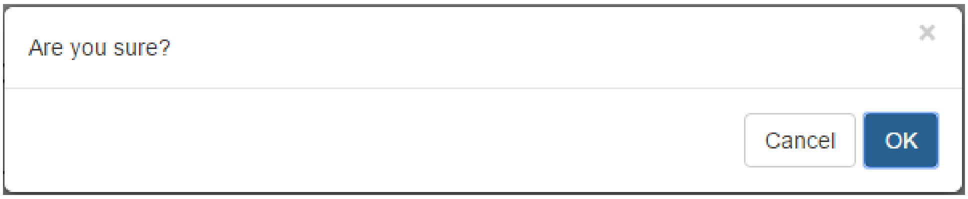
The below is a sample test output when a test case execution times out.

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

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

Modal style dialogs

Flexible Javascript libraries, such as Bootstrap 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();
Thread.sleep(500);
driver.findElement(By.xpath("//div[@class='modal-footer']/button[text()='OK'\
]")).click();
```

Popup Handler Approach

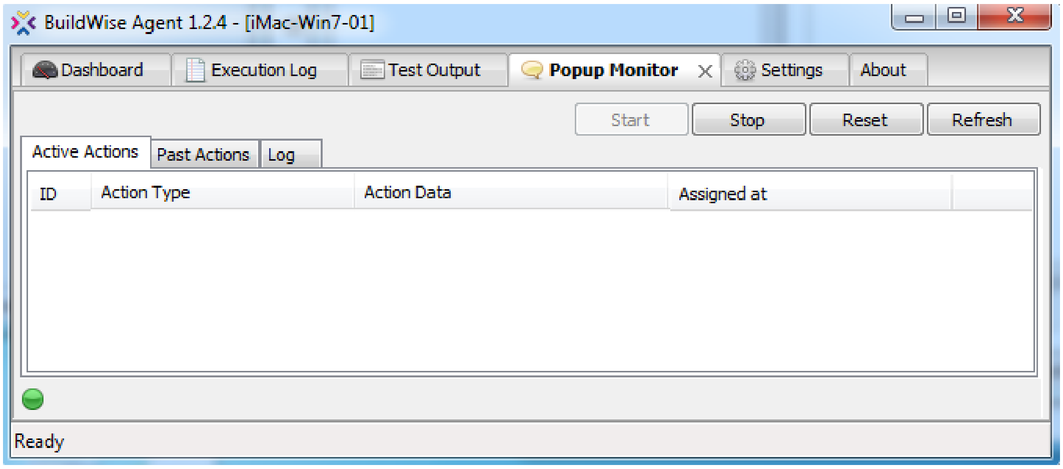
There are other types of pop ups too, such as Basic Authentication and Security warning dialogs. How to handle them? The fundamental difficulty behind pop up dialog handling is that some of these dialogs are native windows, not part of the browser, which means they are beyond the testing library's (i.e. Selenium) control.

Here I introduce a generic approach to handle all sorts of pop up dialogs. Set up a monitoring process (let's call it popup handler) waiting for notifications of possible new pop ups. Once the popup hander receives one, it will try to handle the pop up dialog with data received using windows automation technologies. It works like this:

[2]http://getbootstrap.com/javascript/#modals

```
// ...
NOTIFY_HANDLER_ABOUT_TO_TRIGGER_A_POPUP_OPERATION
PERFORM_OPERATION
// ...
```

BuildWise Agent[3] is a tool for executing automated tests on multiple machines in parallel. It has a free utility named 'Popup handler' just does that.



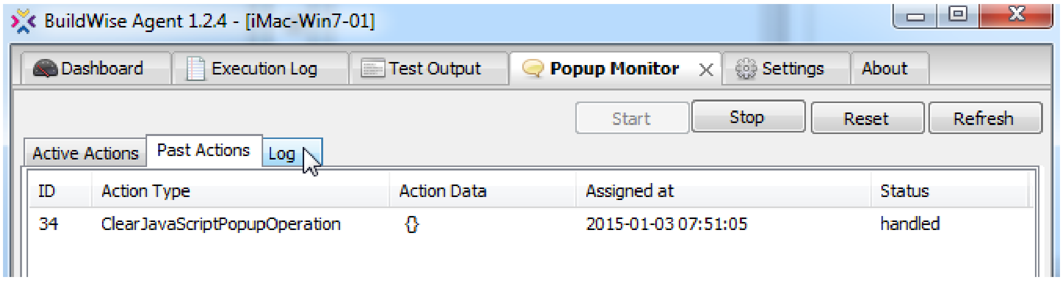
Handle JavaScript dialog with Popup Handler

```
notifyPopupHandlerJavaScript("Message from webpage"); // a helper function
driver.findElement(By.id("buy_now_btn")).click();     // this show a popup
Thread.sleep(15000);  // give some time for handler to handle it
driver.findElement(By.linkText("NetBank")).click();
```

The `notifyPopupHandlerJavaScript` is a function included in the same project.

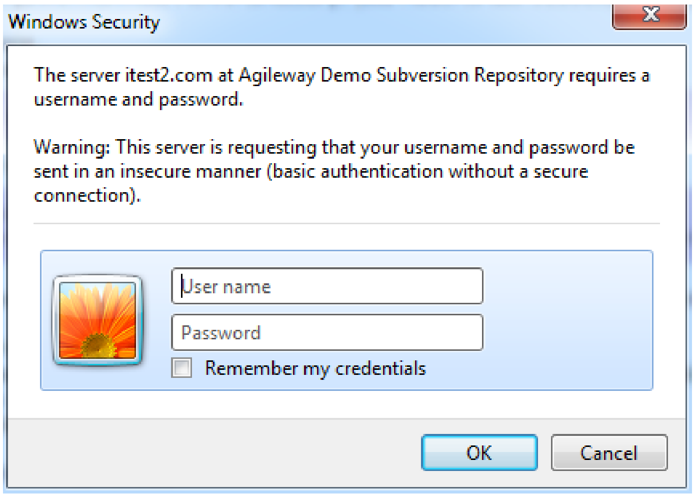
[3] http://testwisely.com/buildwise

```
public static void notifyPopupHandlerJavaScript(String winTitle) throws Exce\
ption {
    String handlerPath = "/popup/win_title=" + URLEncoder.encode(winTitle);
    getUrlText(handlerPath);
}
```



ID	Action Type	Action Data	Assigned at	Status
34	ClearJavaScriptPopupOperation	{}	2015-01-03 07:51:05	handled

Basic or Proxy Authentication Dialog



The server itest2.com at Agileway Demo Subversion Repository requires a username and password.

Warning: This server is requesting that your username and password be sent in an insecure manner (basic authentication without a secure connection).

```
String winTitle = "Windows Security";
String username = "tony";
String password = "password";
notifyPopupHandlerBasicAuth(winTitle, username, password);
driver.get("http://itest2.com/svn-demo/");
Thread.sleep(20000);
driver.findElement(By.linkText("tony/")).click();
```

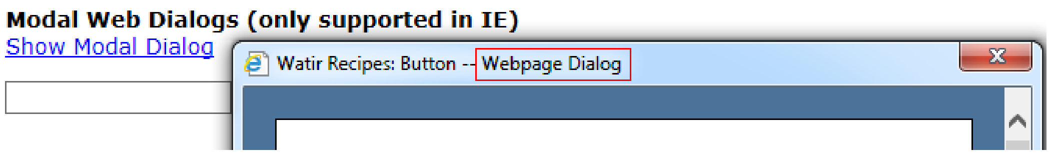
```
public static void notifyPopupHandlerBasicAuth(String winTitle, String usern\
ame, String password) throws Exception {
    String handlerPath = "/basic_authentication/win_title=" + URLEncoder.enc\
ode(winTitle) + "&user=" + username + "&password=" + password;
    getUrlText(handlerPath);
}
```

The same test steps can also be applied to proxy authentication dialogs.

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
Watir Recipes: Button -- Webpage Dialog

HTML Source

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

Test script

```
Set<String> windowHandles = driver.getWindowHandles();
String mainWin = (String) windowHandles.iterator().next();
String modalWin = null;

for (Iterator iter = windowHandles.iterator(); iter.hasNext();) {
  modalWin = (String) iter.next();
}
driver.switchTo().window(modalWin);
driver.findElement(By.name("user")).sendKeys("in-modal");

driver.switchTo().window(mainWin);
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.

Print text for debugging

```
System.out.println("Now on page: " + driver.getTitle());
String app_no = driver.findElement(By.id("app_id")).getText();
System.out.println("Application number is " + app_no);
```

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

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

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.

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.

```
PrintWriter out = new PrintWriter("c:\\temp\\login_page.html");
out.println(driver.getPageSource());
out.close();
```

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

```
WebElement the_element = driver.findElement(By.id("div_parent"));
String the_element_html = (String) ((JavascriptExecutor) driver).executeScri\
pt("return arguments[0].outerHTML;", the_element);
out = new PrintWriter("c:\\temp\\login_parent.xhtml");
out.println(the_element_html);
out.close();
```

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.

```
File scrFile = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(scrFile, new File("c:\\temp\\screenshot.png"));
```

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
File scrFile = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);
String timestamp = new SimpleDateFormat("MMddhhmm").format(new Date());
FileUtils.copyFile(scrFile, new File(TestHelper.tempDir() +  File.separator \
+ "screenshot-" + timestamp + ".png"));
```

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: which test statement (using line number for indication) is failed on and what current web page is like. The first one can be easily found in the testing tools (or command line output). We need the browser to stay open to see the web page after 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 `@After` or `@AfterClass` fixtures like the below:

```
@AfterClass
public static void afterAll() throws Exception {
    driver.quit();
}
```

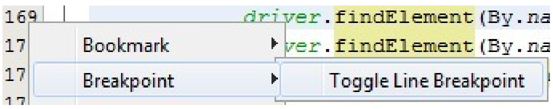
Ideally, we would like to keep the browser open when after running an individual test case and close the browser when running multiple test script files, within the IDE. TestWise[1], an IDE for Selenium Ruby, has this feature. It maybe a possible extension you can add to your Java IDE (such as NetBeans or Eclipse). What I can tell you is that this feature is quite useful.

Debug test execution using Debugger

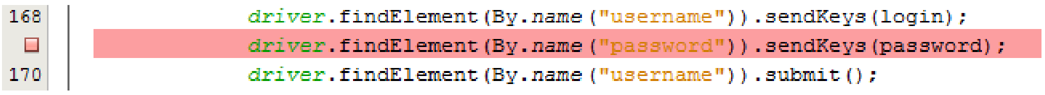
Pause, stop execution and run up to a certain statement are typical debugging features in programming IDEs.

Enable breakpoints

A breakpoint is a stopping or pausing place for debugging purposes. To set a breakpoint, right mouse click the line number and select 'Breakpoint' → 'Toggle Line Breakpoint'.



After set, the statement line where the breakpoint is highlighted.



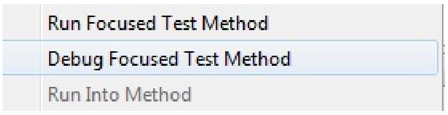
```
driver.findElement(By.name("username")).sendKeys(login);
driver.findElement(By.name("password")).sendKeys(password);
driver.findElement(By.name("username")).submit();
```

You may set more than one breakpoints.

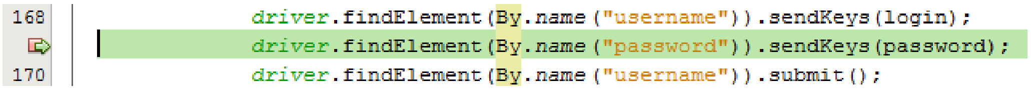
Execute one test case in debugging mode

To start debugging one test case, right mouse click within the lines of the selected test case and select 'Debug Focused Test Method'.

[1]http://testwisely.com/testwise



Test execution starts (will be littler slower in debugging mode),

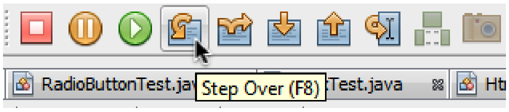


```
                driver.findElement(By.name("username")).sendKeys(login);
                driver.findElement(By.name("password")).sendKeys(password);
                driver.findElement(By.name("username")).submit();
```

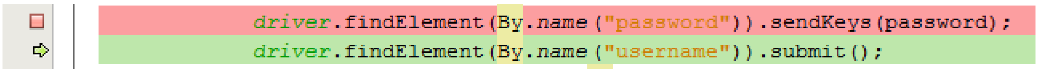
Once the test execution is on halt, you can do inspection against the web page.

Step over test execution

To continue test execution, click the 'Step over' button on the tool bar.



This will execute the just one test statement line, after that, the execution remains in pausing mode again.



```
                driver.findElement(By.name("password")).sendKeys(password);
                driver.findElement(By.name("username")).submit();
```

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.

Get date dynamically

```
// assume today is 2014-12-29
DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd");
String todaysDate = dateFormat.format(new Date());  // => 12/29/2014
driver.findElement(By.name("birth_date")).sendKeys(todaysDate);
```

More flexible way is to use `Calendar`, we can create the following easy to read date related functions (see the helper method in the sample project).

```
today("dd/MM/yyyy"); // => 29/12/2014
tomorrow("AUS"); // => 30/12/2014
yesterday("ISO"); // => 2014-12-28


public static String getDate(String format, int dateDiff) {
    if (format == null) {
        format = "MM/dd/yyyy";
    } else if (format.equals("AUS") || format.equals("UK")) {
        format = "dd/MM/yyyy";
    } else if (format.equals("ISO")) {
        format = "yyyy-MM-dd";
    }
    DateFormat dateFormat = new SimpleDateFormat(format);
    Calendar cal = Calendar.getInstance();
    cal.add(Calendar.DATE, dateDiff);
    return dateFormat.format(cal.getTime());
}
```

```
public static String today(String format) {
    return getDate(format, 0);
}

public static String tomorrow(String format) {
    return getDate(format, 1);
}

public static String yesterday(String format) {
    return getDate(format, -1);
}
```

Example use

```
driver.findElement(By.id("date")).getText().equals(today("MM/dd/yyyy"));
```

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.

```
public static boolean 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.

```
String randomGender = getRandomBoolean() ? "male" : "female";
driver.findElement(By.xpath("//input[@type='radio' and @name='gender' and @v\
alue='" + randomGender + "']")).click();
```

Generate a number within a range

```
// a number within a range
public static int getRandomNumber(int min, int max) {
    return min + (int) (Math.random() * ((max - min) + 1));
}
```

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 99
driver.findElement(By.name("drivers_age")).sendKeys("" +  getRandomNumber(10\
, 99));
```

Get a random character

```
public static char getRandomChar() {
    int rnd = (int) (Math.random() * 52); // or use Random or whatever
    char base = (rnd < 26) ? 'A' : 'a';
    return (char) (base + rnd % 26);
}
```

Get a random string at fixed length

```
public static String getRandomString(int length) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < length; i++) {
        sb.append(getRandomChar());
    }
    return sb.toString();
}


@Test
public void testRandomStringForTextField() throws Exception {
    // ...
    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);
```

Get a random string in a collection

```
public static String getRandomStringIn(String[] array) {
  return array[getRandomNumber(0, array.length - 1)];
}


@Test
public void testRandomStringInCollection() throws Exception {
  // ...
  String[] allowableStrings = new String[]{ "Yes", "No", "Maybe"} ;
  driver.findElement(By.name("username")).sendKeys(getRandomStringIn(allowab\
leStrings ));
}
```

I frequently use this in my test scripts.

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.

```
String outputFilePath = "C:\\temp\\2MB.txt";
FileUtils.writeByteArrayToFile(new File(outputFilePath), new byte[1024 * 102\
4 * 2]);
```

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 sqlite-jdbc[1].

```
String oldestUserLogin = null;

Class.forName("org.sqlite.JDBC");
Connection connection = null;

try {
    connection = DriverManager.getConnection("jdbc:sqlite:C:\\work\\books\\S\
eleniumRecipes-Java\\recipes\\testdata\\sample.db");
    Statement statement = connection.createStatement();
    statement.setQueryTimeout(10);
    ResultSet rs = statement.executeQuery("select login from users order by \
age desc");
    while (rs.next()) {    // read the result set
        oldestUserLogin = rs.getString("login");
        break;
    }
} catch (SQLException e) {
    // probably means no database file is found
    System.err.println(e.getMessage());
} finally {
    try {
        if (connection != null) {
            connection.close();
        }
    } catch (SQLException e) {  // connection close failed.
        System.err.println(e);
    }
```

[1]https://bitbucket.org/xerial/sqlite-jdbc/downloads

```
}

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

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 usage.

Get browser type and version

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

```
driver = new FirefoxDriver();
Capabilities caps = ((RemoteWebDriver) driver).getCapabilities();
String browserName = caps.getBrowserName();
String browserVersion = caps.getVersion();
System.out.println("browserName = " + browserName); // firefox
System.out.println("browserVersion = " + browserVersion); // 28.0
driver.quit();

driver = new ChromeDriver();
caps = ((RemoteWebDriver) driver).getCapabilities();
browserName = caps.getBrowserName();
browserVersion = caps.getVersion();
Platform browserPlatform = caps.getPlatform();
System.out.println("browserName = " + browserName); // chrome
System.out.println("browserVersion = " + browserVersion); // 33.0.1750.152
System.out.println("browserPlatform = " + browserPlatform.toString()); // MAC
driver.quit();

// on Windows platform
// driver = new InternetExplorerDriver();
// browserName will be "internet explorer"
```

Set HTTP Proxy for Browser

Here is an example to set HTTP proxy server for Firefox browser.

```
FirefoxProfile firefoxProfile = new FirefoxProfile();
firefoxProfile.setPreference("network.proxy.type", 1);
// See http://kb.mozillazine.org/Network.proxy.type

firefoxProfile.setPreference("network.proxy.http", "myproxy.com");
firefoxProfile.setPreference("network.proxy.http_port", 3128);
driver = new FirefoxDriver(firefoxProfile);
driver.navigate().to("http://itest2.com/svn-demo/");
```

Verify file download in Chrome

To efficiently verify a file is downloaded, we would like

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

```
Map<String, String> download = new Hashtable<String, String>();
download.put("default_directory", "C:\\temp");
// If Mac or Linux use the path format below
// download.put("default_directory", "/Users/zhimin/tmp");

Map<String, Map> prefs = new Hashtable<String, Map>();
prefs.put("download", download);
DesiredCapabilities capabilities = DesiredCapabilities.chrome();

ChromeOptions chromeOptions = new ChromeOptions();
chromeOptions.setExperimentalOption("prefs", prefs);
driver = new ChromeDriver(chromeOptions);
driver.navigate().to("http://zhimin.com/books/pwta");
driver.findElement(By.linkText("Download")).click();
Thread.sleep(15000); // wait 15 seconds for downloading to complete
```

```
File f = new File("/Users/zhimin/tmp/practical_web_test_automation-sample.pd\
f");
assert f.exists();
```

This is the new way (from v2.37) to pass preferences to Chrome.

More Chrome preferences: http://src.chromium.org/svn/trunk/src/chrome/common/pref_-names.cc[1]

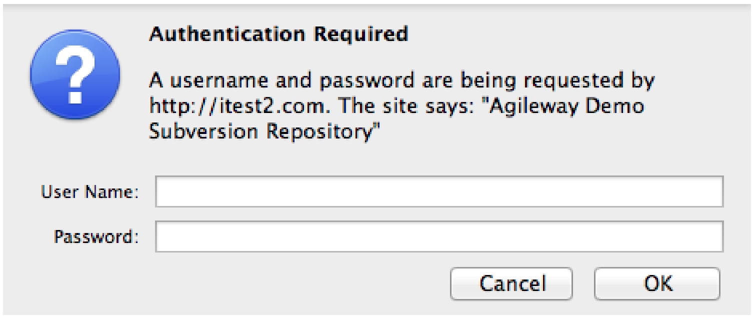
Verify file download in Firefox

```
String myDownloadFolder = "c:\\temp\\";
FirefoxProfile fp = new FirefoxProfile();
fp.setPreference("browser.download.folderList", 2);
fp.setPreference("browser.download.dir", myDownloadFolder);
fp.setPreference("browser.helperApps.neverAsk.saveToDisk", "application/pdf"\
);
// disable Firefox's built-in PDF viewer
fp.setPreference("pdfjs.disabled", true);

driver = new FirefoxDriver(fp);
driver.navigate().to("https://leanpub.com/selenium-recipes-in-java");
driver.findElement(By.linkText("Download")).click();
Thread.sleep(10000); // wait 15 seconds for downloading to complete

File f = new File("C:\\temp\\selenium-recipes-in-java-sample.pdf");
assert f.exists();
```

Bypass basic authentication by embedding username and password in URL

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

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


Authentication Required

A username and password are being requested by http://itest2.com. The site says: "Agileway Demo Subversion Repository"

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

```
driver = new FirefoxDriver();
driver.navigate().to("http://tony:password@itest2.com/svn-demo/");
// got in, click a link
driver.findElement(By.linkText("tony/")).click();
```

Bypass basic authentication with Firefox AutoAuth plugin

There is another complex but quite useful approach to bypass the 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:

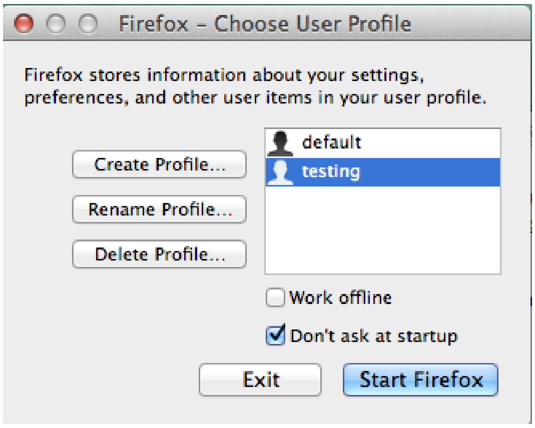
  ```
  "C:\Program Files (x86)\Mozilla Firefox\firefox.exe" -p
  ```

  Mac:

[2]https://addons.mozilla.org/en-US/firefox/addon/autoauth/

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

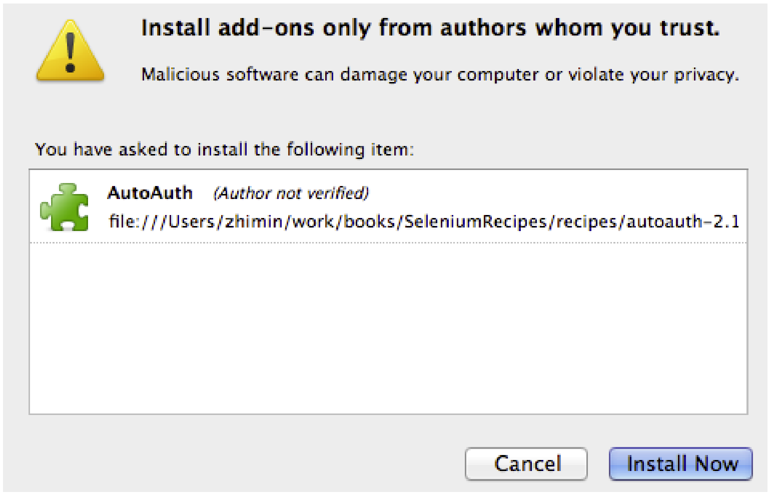
- Create a profile (I name it 'testing') and start Firefox with this profile



Firefox – Choose User Profile

Firefox stores information about your settings, preferences, and other user items in your user profile.

- Install autoauth plugin. A simple way: drag the file *autoauth-2.1-fx+fn.xpi* (included with the test project) to Firefox window.



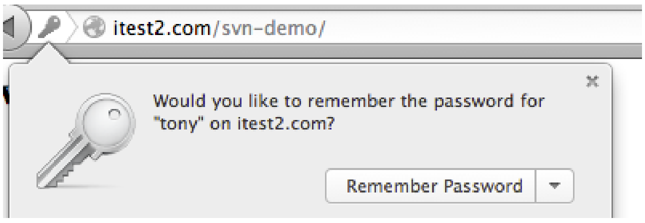
Install add-ons only from authors whom you trust.

Malicious software can damage your computer or violate your privacy.

You have asked to install the following item:

AutoAuth *(Author not verified)*
file:///Users/zhimin/work/books/SeleniumRecipes/recipes/autoauth-2.1

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



Would you like to remember the password for "tony" on itest2.com?

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

```
// Prerequisite: the password is already remembered in 'testing' profile.
File profileDir = new File("/Users/zhimin/Library/Application Support/Firefo\
x/Profiles/tcfyedtq.testing");
FirefoxProfile firefoxProfile = new FirefoxProfile(profileDir);
firefoxProfile.addExtension(new File("/Users/zhimin/work/books/SeleniumRecip\
es-Java/recipes/autoauth-2.1-fx+fn.xpi"));

driver = new FirefoxDriver(firefoxProfile);
driver.navigate().to("http://itest2.com/svn-demo/");
// got in, click a link
driver.findElement(By.linkText("tony/")).click();
```

The hard coded profile path `tcfyedtq.testing` is not ideal, as the test script will fail when running on another machine. The code below will get around that.

```
public static String getFirefoxProfileFolderByName(String profileName) {
    String pathToCurrentUserProfiles = System.getenv("APPDATA") + "\\Mozilla\
\\Firefox\\Profiles"; // Path to profile
    File directory = new File(pathToCurrentUserProfiles);
    File[] fList = directory.listFiles();
    for (File file : fList){
        if (file.isDirectory()) {
            String profileDirPath = file.getAbsolutePath();
            if (profileDirPath.endsWith(profileName)) {
                return profileDirPath;
            }
        }
    }
    return null;
}

// ...
//  in your test case
File profileDir = new File(getFirefoxProfileFolderByName("testing"));
FirefoxProfile firefoxProfile = new FirefoxProfile(profileDir);
```

Manage Cookies

```
driver.navigate().to("http://travel.agileway.net");
driver.manage().addCookie(new Cookie("name", "value"));
Set<Cookie> allCookies = driver.manage().getCookies();
Cookie retrieved = driver.manage().getCookieNamed("name");
assert retrieved.getValue().equals("value");
```

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

- click()
- clickAndHold()
- contextClick()
- doubleClick()
- dragAndDrop()
- dragAndDropBy()
- moveByOffset()
- moveToElement()
- release()

Keyboard interactions

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

The usage

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

Import the classes below in your test scripts to use Actions.

```
import org.openqa.selenium.interactions.Action;
import org.openqa.selenium.interactions.Actions;
```

Check out the Actions API[1] for more.

[1]https://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/interactions/Actions.html

Double click a control

```
WebElement elem = driver.findElement(By.id("pass"));
Actions builder = new Actions(driver);
builder.doubleClick(elem).perform();
```

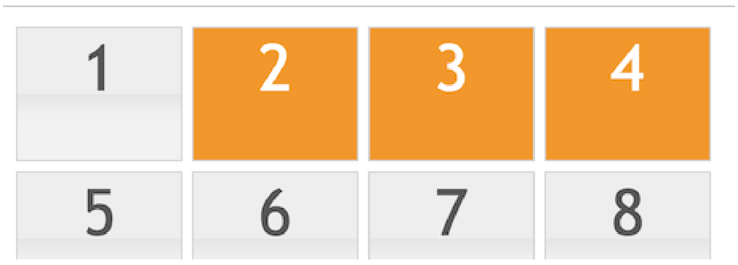
Move mouse to a control - Mouse Over

```
WebElement elem = driver.findElement(By.id("email"));
Actions builder = new Actions(driver);
builder.moveToElement(elem).perform();
```

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();
Thread.sleep(500);
driver.switchTo().frame(0);
List<WebElement> listItems = driver.findElements(By.xpath("//ol[@id='selecta\
ble']/li"));
Actions builder = new Actions(driver);
builder.clickAndHold(listItems.get(1))
       .clickAndHold(listItems.get(3))
       .click()
       .perform();
driver.switchTo().defaultContent();
```

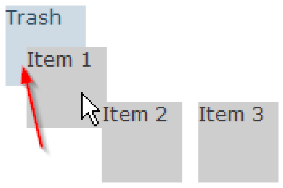


Context Click - right click a control

```
driver.get(TestHelper.siteUrl() +  "text_field.html");
Thread.sleep(500);
WebElement elem = driver.findElement(By.id("pass"));
Capabilities caps = ((RemoteWebDriver) driver).getCapabilities();
String browserName = caps.getBrowserName();
if (caps.getBrowserName() == "firefox") {
    Actions builder = new Actions(driver);
    builder.contextClick(elem)
           .sendKeys(Keys.DOWN)
           .sendKeys(Keys.DOWN)
           .sendKeys(Keys.DOWN)
           .sendKeys(Keys.DOWN)
           .sendKeys(Keys.RETURN)
           .perform();
} else {
    // ...
}
```

Drag and drop

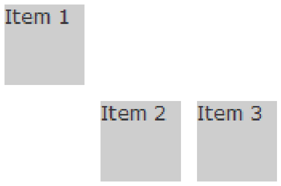
Drag-n-drop is increasingly more common in new web sites. Testing this feature can be largely achieved in Selenium, I used the word ‘largely’ means achieving the same outcome, but not the ‘mouse dragging’ part. For this example page,



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

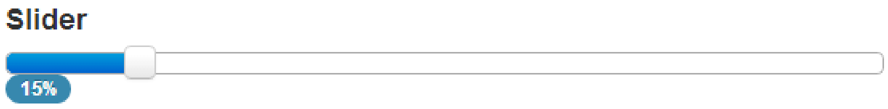
```
driver.get(TestHelper.siteUrl() +  "drag_n_drop.html");
WebElement dragFrom = driver.findElement(By.id("item_1"));
WebElement target = driver.findElement(By.id("trash"));

Actions builder = new Actions(driver);
Action dragAndDrop = builder.clickAndHold(dragFrom)
        .moveToElement(target)
        .release(target)
        .build();

dragAndDrop.perform();
```

The below is a screenshot after the test execution.



Drag slider

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

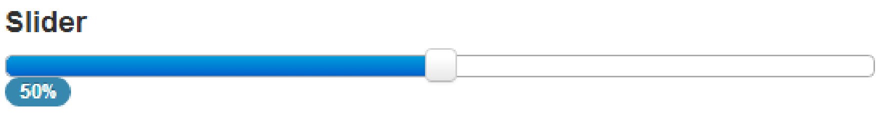


The test below simulates 'dragging the slider to the right'.

```
assert "15%".equals(driver.findElement(By.id("pass_rate")).getText());
WebElement elem = driver.findElement(By.id("pass-rate-slider"));

Actions move = new Actions(driver);
Action action = (Action) move.dragAndDropBy(elem, 30, 0).build();
action.perform();

assert !"15%".equals(driver.findElement(By.id("pass_rate")).getText());
```

More information about `drag_and_drop_by` can be found at Selenium WebDriver Actions API[2].

The below is a screenshot after the test execution.



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

Send key sequences - Select All and Delete

```
driver.get(TestHelper.siteUrl() +  "text_field.html");
driver.findElement(By.id("comments")).sendKeys("Multiple Line\r\n Text");
WebElement elem = driver.findElement(By.id("comments"));

Actions builder = new Actions(driver);
builder.click(elem)
       .keyDown(Keys.CONTROL)
       .sendKeys("a")
       .keyUp(Keys.CONTROL)
       .perform();
// this different from click element, the key is send to browser directly
builder = new Actions(driver);
builder.sendKeys(Keys.BACK_SPACE).perform()
```

[2]https://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/interactions/Actions.html

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

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.

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.

HTML Source

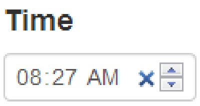
```
<input id="email" name="email" type="email" style="height:30px; width: 280px\
;">
```

Test Script

```
driver.findElement(By.id("email")).sendKeys("test@wisely.com");
```

HTML5 Time Field

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



HTML Source

```
<input id="start_time_1" name="start_time" type="time" style="height:30px; w\
idth: 120px;">
```

The test scripts below do the following:

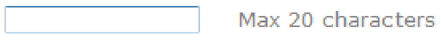
1. make sure the focus is not on this time field control
2. click and focus the time field
3. clear existing time
4. enter a new time

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

// focus on another ...
driver.findElement(By.id("home_link")).sendKeys("");
Thread.sleep(500);

// now back to change it
driver.findElement(By.id("start_time")).click();
// [:delete, :left, :delete, :left, :delete]
driver.findElement(By.id("start_time")).sendKeys(Keys.DELETE);
driver.findElement(By.id("start_time")).sendKeys(Keys.LEFT);
driver.findElement(By.id("start_time")).sendKeys(Keys.DELETE);
driver.findElement(By.id("start_time")).sendKeys(Keys.LEFT);
driver.findElement(By.id("start_time")).sendKeys(Keys.DELETE);

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

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 c\
haracters</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().equals("Max 20 characters");
```

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")).clear();
driver.findElement(By.name("person_name")).sendKeys("Wise Tester too");
((JavascriptExecutor) driver).executeScript("$('#person_name_textbox').trigg\
er('change')");
assert driver.findElement(By.id("person_name_label")).getText().equals("Wise\
 Tester too");
```

Scroll to the bottom of a page

Calling JavaScript API.

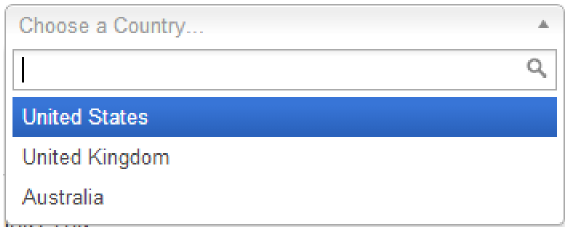
```
((JavascriptExecutor) driver).executeScript("window.scrollTo(0, document.bod\
y.scrollHeight);");
```

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

```
driver.findElement(By.tagName("body")).sendKeys(Keys.chord(Keys.CONTROL, Key\
s.END));
```

Chosen - Standard Select

Chosen 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="chosen_single" class="chosen-select"  data-placeholder="Choose a\
 Country..." style="width:350px;">
  <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 `chosen-select`. 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

```
<div class="chosen-container chosen-container-single chosen-container-active\
" style="width: 350px;" title="" id="chosen_single_chosen">
  <a class="chosen-single chosen-default" tabindex="-1"><span>Choose a Count\
ry...</span><div><b></b></div></a>
  <div class="chosen-drop">
    <div class="chosen-search">
      <input type="text" autocomplete="off" tabindex="2">
    </div>
    <ul class="chosen-results">
      <li class="active-result" style="" data-option-array-index="1">United \
States</li>
      <li class="active-result result-selected" style="" data-option-array-i\
ndex="2">United Kingdom</li>
      <li class="active-result" style="" data-option-array-index="3">Austral\
ia</li>
    </ul>
  </div>
</div>
```

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 Chosen emulates the standard select list first. In Chosen, clicking the 'Choose a Country' is actually clicking a hyper link with class "chosen-single" under the div with ID "*chosen_-single_chosen*" (the ID is whatever set in the select element, followed by '_chosen'); selecting an option is clicking an list item (tag: `li`) with class 'active-result'. With that knowledge, plus XPath in Selenium, we can drive a Chosen standard select box with the test scripts below:

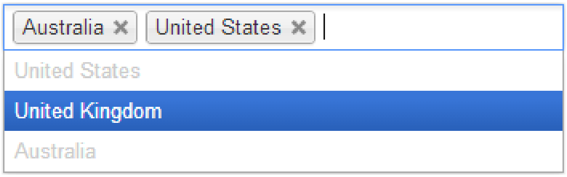
```
Thread.sleep(2000);  // wait enough time to load JS
driver.findElement(By.xpath("//div[@id='chosen_single_chosen']//a[contains(@\
class,'chosen-single')]")).click();
List<WebElement> available_items = driver.findElements(By.xpath("//div[@id='\
chosen_single_chosen']//div[@class='chosen-drop']//li[contains(@class,'activ\
e-result')]"));
for (WebElement item : available_items) {
    if (item.getText().equals("Australia")) {
        item.click();
        break;
    }
}
Thread.sleep(1000);

driver.findElement(By.xpath("//div[@id='chosen_single_chosen']//a[contains(@\
class,'chosen-single')]")).click();
available_items = driver.findElements(By.xpath("//div[@id='chosen_single_cho\
sen']//div[@class='chosen-drop']//li[contains(@class,'active-result')]"));
for (WebElement item : available_items) {
    if (item.getText().equals("United States")) {
        item.click();
        break;
    }
}
```

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

```
Thread.sleep(1000);
driver.findElement(By.xpath("//div[@id='chosen_single_chosen']//a[contains(@\
class,'chosen-single')]")).click();

WebElement search_text_field = driver.findElement(By.xpath("//div[@id='chose\
n_single_chosen']//div[@class='chosen-drop']//div[contains(@class,'chosen-se\
arch')]/input"));
search_text_field.sendKeys("United King");
Thread.sleep(500); // let filtering finishing
// select first selected option
search_text_field.sendKeys(Keys.ENTER);
```

Chosen - Multiple Select

Chosen[1] also enhances the multiple selection (a lot).



HTML Source

```
<select id="chosen_multiple" class="chosen-select" multiple data-placeholder\
="Choose a Country..."  style="width:350px;">
  <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 in HTML from the standard multiple select list is the class 'chosen-select'.

Generated HTML Source

[1]http://harvesthq.github.io/chosen/

```
<div class="chosen-container chosen-container-multi chosen-container-active"\
 style="width: 350px;" title="" id="chosen_multiple_chosen">
  <ul class="chosen-choices">
    <li class="search-choice"><span>Australia</span><a class="search-choice-\
close" data-option-array-index="3"></a></li>
    <li class="search-choice"><span>United States</span><a class="search-cho\
ice-close" data-option-array-index="1"></a></li>
    <li class="search-field"><input type="text" value="Choose a Country..." \
class="" autocomplete="off" style="width: 25px;" tabindex="4"></li>
  </ul>
  <div class="chosen-drop">
    <ul class="chosen-results">
      <li class="result-selected" style="" data-option-array-index="1">Unite\
d States</li>
      <li class="active-result" style="" data-option-array-index="2">United \
Kingdom</li>
      <li class="result-selected" style="" data-option-array-index="3">Austr\
alia</li>
      </ul>
  </div>
</div>
```

Astute readers will find the generated HTML fragment is quite 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.

```
Thread.sleep(2000);  // wait enough time to load JS
driver.findElement(By.xpath("//div[@id='chosen_multiple_chosen']//li[@class=\
'search-field']/input")).click();
List<WebElement> available_items = driver.findElements(By.xpath("//div[@id='\
chosen_multiple_chosen']//div[@class='chosen-drop']//li[contains(@class,'act\
ive-result')]"));
for (WebElement item : available_items) {
    if (item.getText().equals("Australia")) {
        item.click();
        break;
    }
}
```

```
Thread.sleep(1000);

// select another
driver.findElement(By.xpath("//div[@id='chosen_multiple_chosen']//li[@class=\
'search-field']/input")).click();
available_items = driver.findElements(By.xpath("//div[@id='chosen_multiple_c\
hosen']//div[@class='chosen-drop']//li[contains(@class,'active-result')]"));
for (WebElement item : available_items) {
    if (item.getText().equals("United Kingdom")) {
        item.click();
        break;
    }
}
```

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.

```
Thread.sleep(500);
// clear all selections
List<WebElement> closeButtons = driver.findElements(By.xpath("//div[@id='cho\
sen_multiple_chosen']//ul[@class='chosen-choices']/li[contains(@class,'searc\
h-choice')]/a[contains(@class,'search-choice-close')]"));
for (WebElement closeButton : closeButtons) {
    closeButton.click();
}

// select one after clear
driver.findElement(By.xpath("//div[@id='chosen_multiple_chosen']//li[@class=\
'search-field']/input")).click();
available_items = driver.findElements(By.xpath("//div[@id='chosen_multiple_c\
hosen']//div[@class='chosen-drop']//li[contains(@class,'active-result')]"));
for (WebElement item : available_items) {
    if (item.getText().equals("United States")) {
        item.click();
        break;
    }
}
```

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:

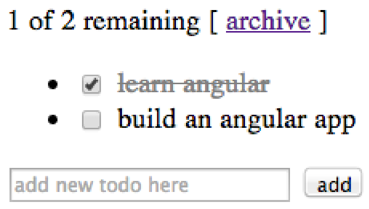
```
public void clearChosen(String elemId) throws Exception {
    Thread.sleep(500);
    List<WebElement> closeButtons = driver.findElements(By.xpath("//div[@id=\
'" + elemId + "']//ul[@class='chosen-choices']/li[contains(@class,'search-ch\
oice')]/a[contains(@class,'search-choice-close')]"));
    for (WebElement closeButton : closeButtons) {
        closeButton.click();
    }
}

public void selectChosenByLabel(String elemId, String label) {
    driver.findElement(By.xpath("//div[@id='" + elemId + "']//li[@class='sea\
rch-field']/input")).click();
    List<WebElement> availableItems = driver.findElements(By.xpath("//div[@i\
d='" + elemId + "']//div[@class='chosen-drop']//li[contains(@class,'active-r\
esult')]"));
    for (WebElement item : availableItems) {
        if (item.getText().equals(label)) {
            item.click();
            break;
        }
    }
}

@Test
public void testChosenMultipleCallingMethods() throws Exception {
    // ... land to the page with a chosen select list
    clearChosen("chosen_multiple_chosen");
    selectChosenByLabel("chosen_multiple_chosen", "United States");
    selectChosenByLabel("chosen_multiple_chosen", "Australia");
}
```

You can find more techniques for writing maintainable tests from my other book *Practical*

Web Test Automation[2].

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

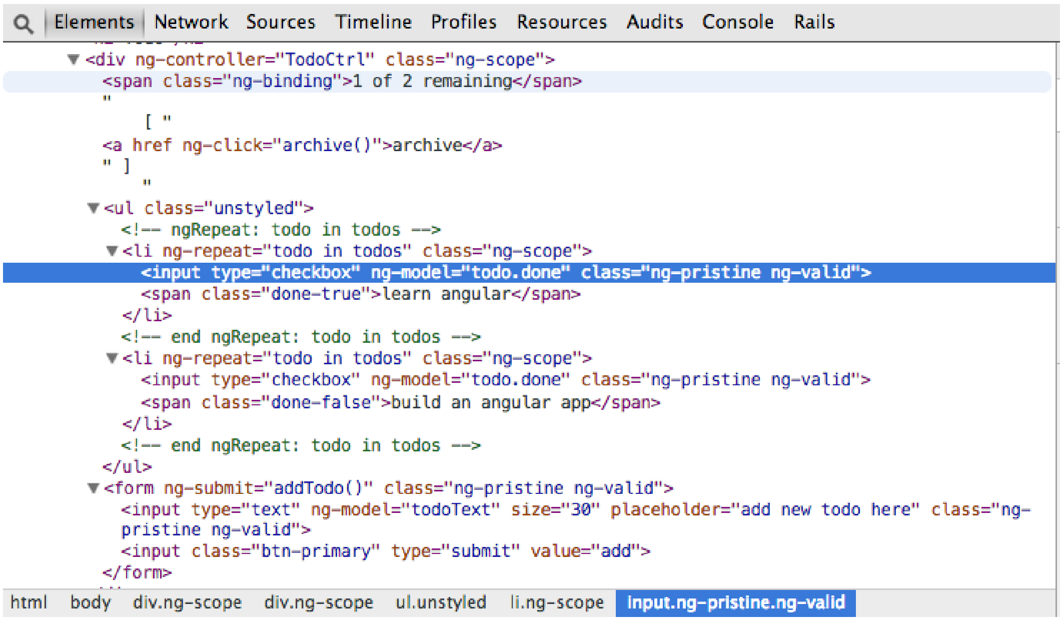
The page source (via "View Page Source" in browser) is different from what you saw on the page. It contains some kind of dynamic coding (*ng-xxx*).

```
<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"
           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".

[2]https://leanpub.com/practical-web-test-automation

Browser inspect view



```
▼<div ng-controller="TodoCtrl" class="ng-scope">
   <span class="ng-binding">1 of 2 remaining</span>
   "
       [ "
   <a href ng-click="archive()">archive</a>
   " ]
       "
  ▼<ul class="unstyled">
     <!-- ngRepeat: todo in todos -->
    ▼<li ng-repeat="todo in todos" class="ng-scope">
       <input type="checkbox" ng-model="todo.done" class="ng-pristine ng-valid">
       <span class="done-true">learn angular</span>
     </li>
     <!-- end ngRepeat: todo in todos -->
    ▼<li ng-repeat="todo in todos" class="ng-scope">
       <input type="checkbox" ng-model="todo.done" class="ng-pristine ng-valid">
       <span class="done-false">build an angular app</span>
     </li>
     <!-- end ngRepeat: todo in todos -->
   </ul>
  ▼<form ng-submit="addTodo()" class="ng-pristine ng-valid">
     <input type="text" ng-model="todoText" size="30" placeholder="add new todo here" class="ng-
     pristine ng-valid">
     <input class="btn-primary" type="submit" value="add">
   </form>
```

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

```
assert driver.getPageSource().contains("1 of 2 remaining");
driver.findElement(By.xpath("//input[@ng-model='todoText']")).sendKeys("Lear\
n test automation");
driver.findElement(By.xpath("//input[@type = 'submit' and @value='add']")).c\
lick();
Thread.sleep(500);
driver.findElements(By.xpath("//input[@type = 'checkbox' and @ng-model='todo\
.done']")).get(2).click();
Thread.sleep(1000);
assert driver.getPageSource().contains("1 of 3 remaining");
```

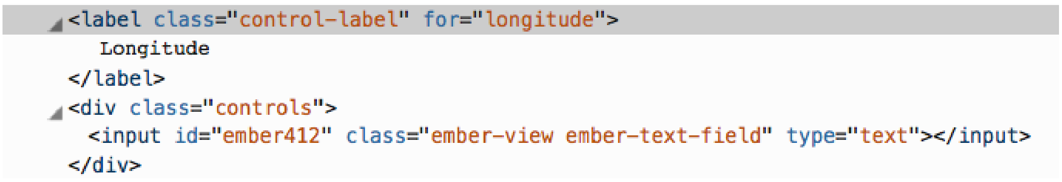
Ember JS web pages

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

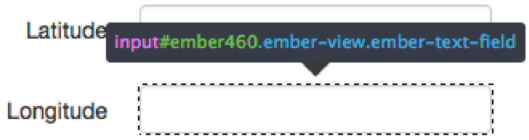
```
<div class="control-group">
  <label class="control-label" for="longitude">Longitude</label>
  <div class="controls">
    {{view Ember.TextField valueBinding="longitude"}}
  </div>
</div>
```

Browser inspect view

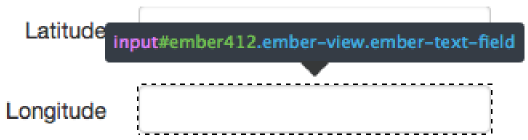


```
<label class="control-label" for="longitude">
   Longitude
</label>
<div class="controls">
  <input id="ember412" class="ember-view ember-text-field" type="text"></input>
</div>
```

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.

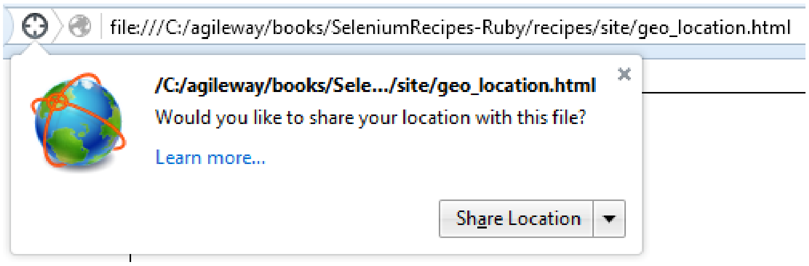


So we shall use another way to identify the element.

```
List<WebElement> ember_text_fields = driver.findElements(By.xpath("//div[@cl\
ass='controls']/input[@class='ember-view ember-text-field']"));
ember_text_fields.get(0).sendKeys("-24.0034583945");
ember_text_fields.get(1).sendKeys("146.903459345");
ember_text_fields.get(2).sendKeys("90%");

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

"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.

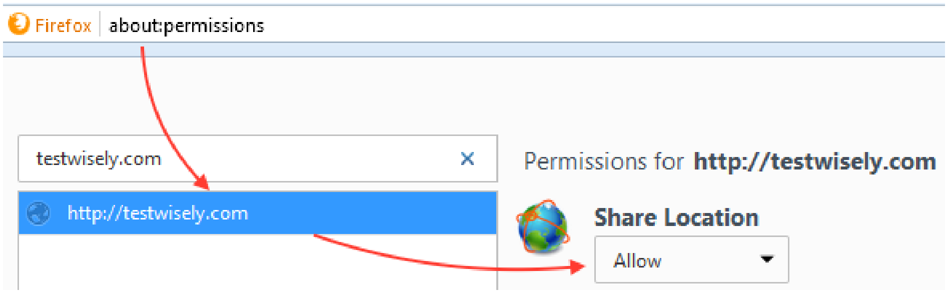

file:///C:/agileway/books/SeleniumRecipes-Ruby/recipes/site/geo_location.html

/C:/agileway/books/Sele.../site/geo_location.html

Would you like to share your location with this file?

This is a native popup window, in other words, Selenium WebDriver cannot drive it. There is a workaround though, that is pre-allowing "Share Location" for a specific website for a browser profile. Here are the steps for Firefox.

1. Open Firefox with a specific profile for testing
2. Open the site
3. Type `about:permissions` in the address
4. Select the site and choose "Allow" option for "Share Location"



Permissions for **http://testwisely.com**

Share Location: Allow

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

```
File profileDir = new File(getFirefoxProfileFolderByName("testing"));
FirefoxProfile firefoxProfile = new FirefoxProfile(profileDir);
driver = new FirefoxDriver(firefoxProfile)

driver.findElement(By.id("use_current_location_btn")).click();
Thread.sleep(5000);
assert driver.findElement(By.id("demo")).getText().contains("Latitude:");
```

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.

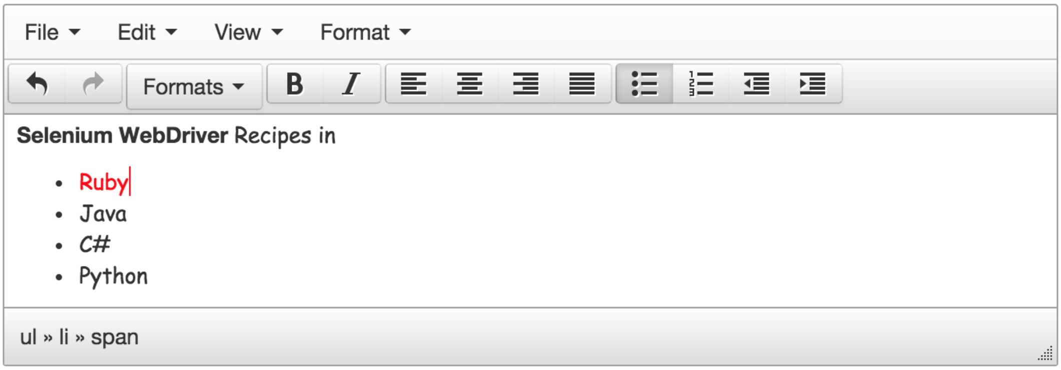
```
String lati = "-34.915379";
String longti = "138.576777";
String js = "window.navigator.geolocation.getCurrentPosition=function(succes\
s){;  var position = {'coords' : {'latitude': '" + lati + "','longitude': '"\
 + longti +  "'}}; success(position);}";
((JavascriptExecutor) driver).executeScript(js);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

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.

TinyMCE

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



Selenium WebDriver Recipes in

- Ruby
- Java
- C#
- Python

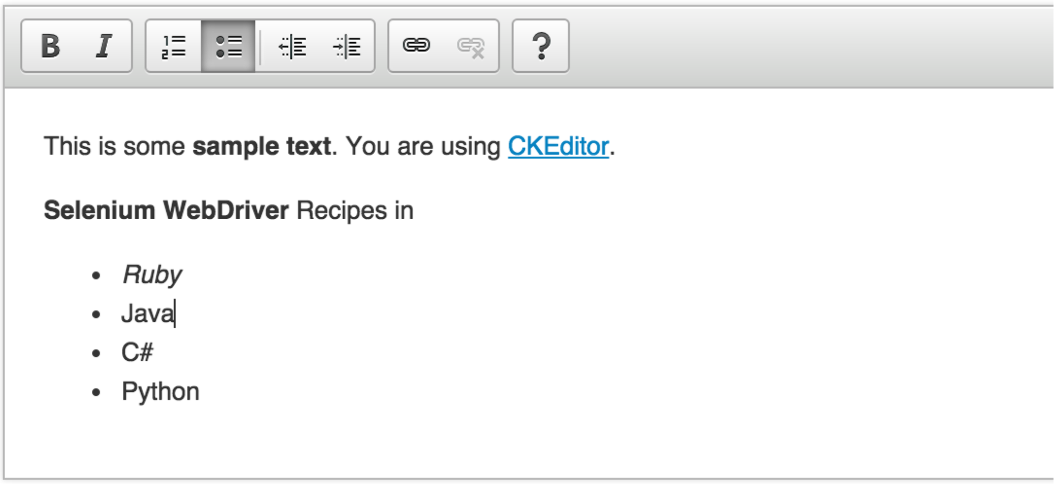
The rich text is rendered inside an inline frame within TinyMCE. To test it, we need to "switch to" that frame.

[1]http://www.tinymce.com/enterprise/using.php

```
driver.get(TestHelper.siteUrl() + "tinymce-4.1.9/tinyice_demo.html");

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

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

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

// Insert text by calling JavaScripts
((JavascriptExecutor) driver).executeScript("tinyMCE.activeEditor.insertCont\
ent('<p>Brisbane</p>')");
```

CKEditor

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


This is some **sample text**. You are using CKEditor.

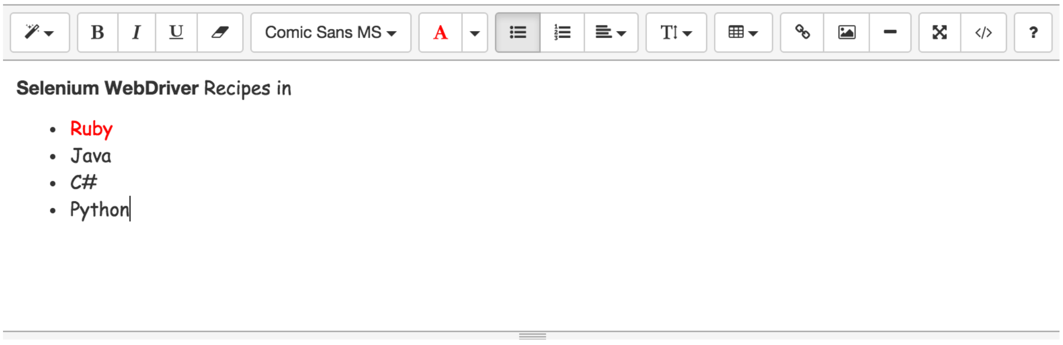
Selenium WebDriver Recipes in

- *Ruby*
- Java
- C#
- Python

```
driver.get(TestHelper.siteUrl()  + "ckeditor-4.4.7/samples/uicolor.html");

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

// Clear content another Method:  Using ActionBuilder
Actions builder = new Actions(driver);
builder.click(editorBody)
       .keyDown(Keys.CONTROL)
       .sendKeys("a")
       .keyUp(Keys.CONTROL)
       .perform();

builder.sendKeys(Keys.BACK_SPACE)
       .perform();

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

SummerNote

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



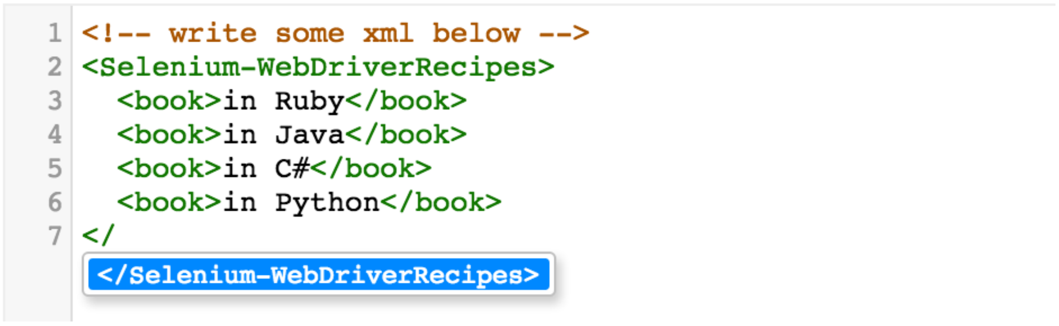
Selenium WebDriver Recipes in

- Ruby
- Java
- *C#*
- Python

```
driver.get(TestHelper.siteUrl() + "summernote-0.6.3/demo.html");
Thread.sleep(500);
driver.findElement(By.xpath("//div[@class='note-editor']/div[@class='note-ed\
itable']")).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>");
```

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.



```
<!-- write some xml below -->
<Selenium-WebDriverRecipes>
  <book>in Ruby</book>
  <book>in Java</book>
  <book>in C#</book>
  <book>in Python</book>
</
```

```java
driver.get(TestHelper.siteUrl() + "codemirror-5.1/demo/xmlcomplete.html");
WebElement elem = driver.findElement(By.className("CodeMirror-scroll"));
elem.click();
Thread.sleep(500);
// elem.sendKeys does not work
Actions builder = new Actions(driver);
builder.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.

Raise exceptions to fail test

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

```
if (!System.getProperty("os.name").equals("Windows 7")) {
    throw new RuntimeException("Unsupported platform: " + osPlatform);
}
```

In test output (when running on Windows):

```
java.lang.RuntimeException: Unsupported platform: Windows 8
        at ProgrammingTest.TestThrowNewExceptions
```

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 anther more complete example.

```
try {
  driver =  new ChromeDriver();
  // ...
} catch (Exception ex) {
  System.out.println("Error occurred: " + ex);
  ex.printStackTrace();
} finally {
  driver.quit();
}
```

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

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
   public void testFoo() throws Exception {
     // TODO
   }
   ```

   I like this way better.

   ```
   @Test
   public void testFoo() throws Exception {
     throw new RuntimeException("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 new RuntimeException("Stop here, I take over from now. I delete this l\
    ater.")
```

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:

```
try {
    driver.findElement(By.name("notExists")).click();
} catch (Exception ex) {
    System.out.println("Error occurred:  " + ex + ", but it is OK to ignore"\
);
}
// ...
```

Read external file

We can use Java'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:

```
String filePath = "C:\\testdata\\in.xml";  // bad
File file = new File(filePath);
// ...
```

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).

```
/* in a helper file */
public static String scriptDir() {
  // you may use environment variables to make the path dynamically settable
  if (isWindows()) {
      return "C:\\agileway\\books\\SeleniumRecipes-Java\\recipes";
  } else if (isMac()) {
      return "/Users/zhimin/work/books/SeleniumRecipes-Java/recipes";
  } else if (isUnix()) {
      return "/home/zhimin/work/books/SeleniumRecipes-Java/recipes";
  } else {
      throw new RuntimeException("Your OS is not support!!");
  }
}

/* in test file */
@Test
public void testReadExtnernalFile() throws Exception {
    String filePath = TestHelper.scriptDir() +  File.separator + "testdata" \
+   File.separator + "in.xml";
    File file = new File(filePath);
    assert file.exists();
    String content = FileUtils.readFileToString(file);
    System.out.println("content = " + content);
}
```

By using `File.separator`, the test script can work on both Windows and Unix platforms.

Data-Driven Tests with Excel

Data-Driven Testing means a test's input are driven from external sources, quite commonly in Excel or CSV files. For instance, if there is a list of user credentials with different roles and the login process is the same (but with different assertions), you can extract the test data from an excel spreadsheet and execute it one by one.

A sample spreadsheet (*users.xls*) contains three username-password combination:

DESCRIPTION	LOGIN	PASSWORD	EXPECTED_TEXT
Valid Login	agileway	test	Login successful!
User name not exists	notexists	smartass	Login is not valid
Password not match	agileway	badpass	Password is not valid

The test scripts below reads the above and uses the login data to drive the browser to perform tests, using Apache POI[1] library.

```
String filePath = TestHelper.scriptDir() + File.separator + "testdata" + Fil\
e.separator + "users.xls";
try {
    POIFSFileSystem fs = new POIFSFileSystem(new FileInputStream(filePath));
    HSSFWorkbook wb = new HSSFWorkbook(fs);
    HSSFSheet sheet = wb.getSheetAt(0);
    HSSFRow row;
    HSSFCell cell;

    int rows;     // number of rows
    rows = sheet.getPhysicalNumberOfRows();

    int cols = 0; // number of columns
    int tmp = 0;

    System.out.println("rows = " + rows);
    // This trick ensures that we get the data properly
    // even if it doesn't start from first few rows
    for (int i = 0; i < rows; i++) {
        row = sheet.getRow(i);
        if (row != null) {
            tmp = sheet.getRow(i).getPhysicalNumberOfCells();
            if (tmp > cols) {
                cols = tmp;
            }
        }
    }

    // starting from row 1, ignore the header row
```

[1]http://poi.apache.org/index.html

```
    for (int r = 1; r < rows; r++) {
        row = sheet.getRow(r);
        if (row != null) {

            String description = row.getCell(0).getStringCellValue();
            String login = row.getCell(1).getStringCellValue();
            String password = row.getCell(2).getStringCellValue();
            String expectedText = row.getCell(3).getStringCellValue();

            driver.get("http://travel.agileway.net");
            driver.findElement(By.name("username")).sendKeys(login);
            driver.findElement(By.name("password")).sendKeys(password);
            driver.findElement(By.name("username")).submit();
            assert driver.findElement(By.tagName("body")).getText().contains\
(expectedText);

            try {
                // if logged in OK, try log out, so next one can continue
                driver.findElement(By.linkText("Sign off")).click();
            } catch (Exception ex) {
                // ignore
            }
        }
    }
} catch (Exception ioe) {
    ioe.printStackTrace();
}
```

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 OpenCSV[2] library:

[2]http://opencsv.sourceforge.net/

```
String csvFilePath = TestHelper.scriptDir() + File.separator + "testdata" + \
File.separator + "users.csv";
CSVReader reader = new CSVReader(new FileReader(csvFilePath));
String[] nextLine;
while ((nextLine = reader.readNext()) != null) {
    // nextLine[] is an array of values from the line
    String login = nextLine[1];
    String password = nextLine[2];
    String expectedText = nextLine[3];

    if (login.equals("LOGIN")) { // header row
        continue;
    }
    driver.get("http://travel.agileway.net");
    driver.findElement(By.name("username")).sendKeys(login);
    driver.findElement(By.name("password")).sendKeys(password);
    driver.findElement(By.name("username")).submit();
    assert driver.findElement(By.tagName("body")).getText().contains(expecte\
dText);

    try {
        // if logged in OK, try log out, so next one can continue
        driver.findElement(By.linkText("Sign off")).click();
    } catch (Exception ex) {
        // ignore
    }
}
```

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).

```
WebElement elemByName = driver.findElement(By.name("ctl00$m$g_dcb0d043_e7f0_\
4128_99c6_71c113f45dd8$ctl00$tAppName"));
elemByName.sendKeys("full name");
elemByName.clear();
Thread.sleep(1000);
driver.findElement(By.xpath("//input[contains(@name, 'tAppName')]")).sendKey\
s("I still can");
```

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.

```
WebElement elem = driver.findElement(By.id("user"));
elem.sendKeys("agileway");
Thread.sleep(1000); // sleep for seeing the effect

elem.sendKeys(Keys.chord(Keys.CONTROL, "a"));
elem.sendKeys(Keys.BACK_SPACE);
Thread.sleep(1000);
elem.sendKeys("testwisely");
Thread.sleep(1000);
elem.sendKeys(Keys.ENTER);  //submit the form
```

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

The full list can be found at org.openqa.selenium.Keys documentation[3].

Use of Unicode in test scripts

Selenium WebDriver does support Unicode.

```
String text = driver.findElement(By.id("unicode_test")).getText();
assert "□□".equals(readString);
// if does not work, try below
// text = new String(text.getBytes(Charset.forName("utf-8")));

String sentStr = "проворный";
driver.findElement(By.id("user")).sendKeys(sentStr);
```

If you get the following error when compiling,

```
 error: unmappable character for encoding Cp1252
```

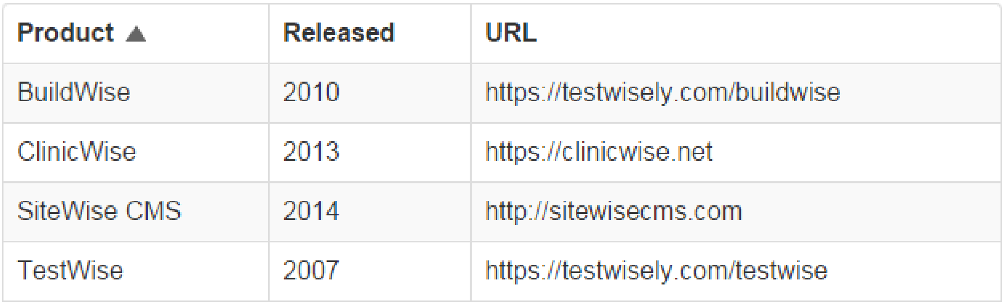
Adding `encoding="UTF-8"` to your `javac` task in Ant's build.xml will solve it.

[3]https://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/Keys.html

```
<javac encoding="UTF-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.



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.

```
driver.findElement(By.id("heading_product")).click(); // sort asc
List<WebElement> firstCells = driver.findElements(By.xpath("//tbody/tr/td[1]\
"));
ArrayList<String> product_names = new ArrayList(CollectionUtils.collect(firs\
tCells, new Transformer() {
    public Object transform(Object input) {
        return ((WebElement)input).getText();
    }
}));

ArrayList<String> sortedProductNames = (ArrayList<String>) product_names.clo\
ne();
Collections.sort(sortedProductNames);
assert product_names.equals(sortedProductNames);

driver.findElement(By.id("heading_product")).click(); // sort desc
Thread.sleep(500);
```

```
firstCells = driver.findElements(By.xpath("//tbody/tr/td[1]"));
product_names = new ArrayList(CollectionUtils.collect(firstCells, new Transf\
ormer() {
    public Object transform(Object input) {
        return ((WebElement)input).getText();
    }
}));
sortedProductNames = (ArrayList<String>) product_names.clone();
Collections.sort(sortedProductNames, Collections.reverseOrder());
assert product_names.equals(sortedProductNames);
```

This approach is not limited to data in tables. The below script extracts the scores from the elements like `<span class='score'>98</span>`.

```
List<WebElement> scoreElems = driver.findElements(By.xpath("//div[@id='resul\
ts']//span[@class='score']"));
// ...
```

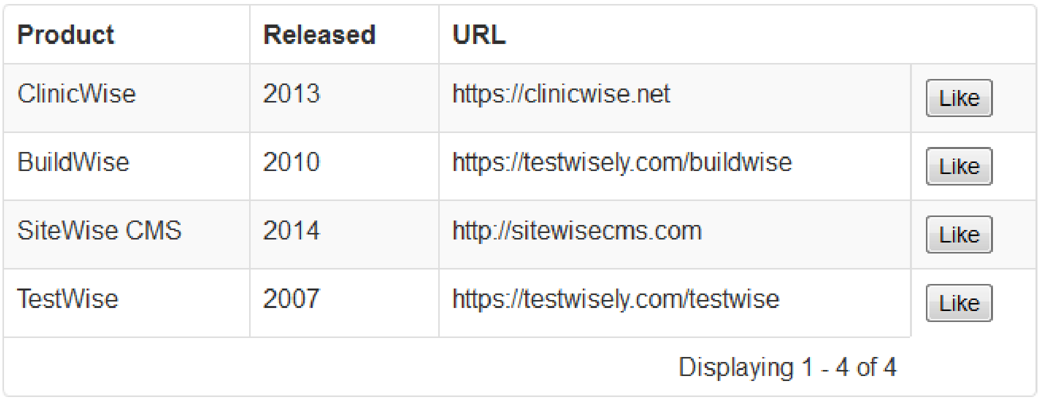
Verify uniqueness of a set of data

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.

```
List<WebElement> secondCells = driver.findElements(By.xpath("//tbody/tr/td[2\
]"));
ArrayList<String> yearsReleased = new ArrayList(CollectionUtils.collect(seco\
ndCells, new Transformer() {
    public Object transform(Object input) {
        return ((WebElement)input).getText();
    }
}));
Set<String> uniqYearsReleased = new HashSet<>(yearsReleased);
assert yearsReleased.size() == uniqYearsReleased.size();
```

Extract dynamic visible data rows from a results table

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



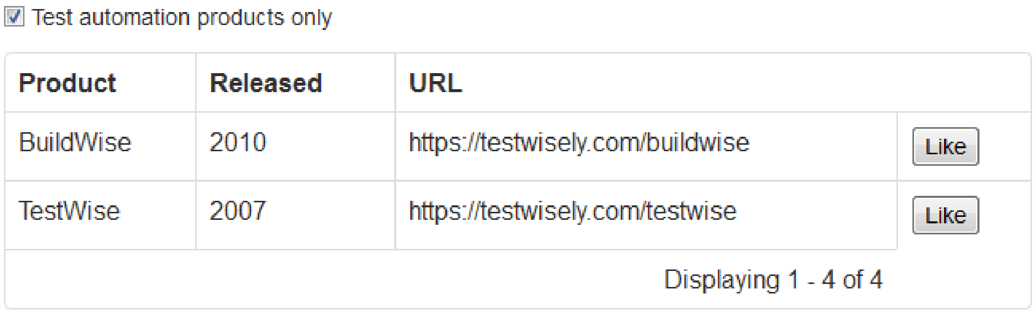
Product	Released	URL	
ClinicWise	2013	https://clinicwise.net	Like
BuildWise	2010	https://testwisely.com/buildwise	Like
SiteWise CMS	2014	http://sitewisecms.com	Like
TestWise	2007	https://testwisely.com/testwise	Like

Displaying 1 - 4 of 4

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

```
driver.navigate().to(TestHelper.siteUrl() + "data_grid.html");
driver.manage().window().setSize(new Dimension(1024, 768));

List<WebElement>  rows = driver.findElements(By.xpath("//table[@id='grid']/t\
body/tr"));
assert(4 == rows.size());
String firstProductName = driver.findElement(By.xpath("//table[@id='grid']//\
tbody/tr[1]/td[1]")).getText();
assert("ClinicWise".equals(firstProductName));
driver.findElement(By.xpath("//table[@id='grid']//tbody/tr[1]/td/button")).c\
lick();
```

Now check "Test automation products only" checkbox, and only two products are shown.



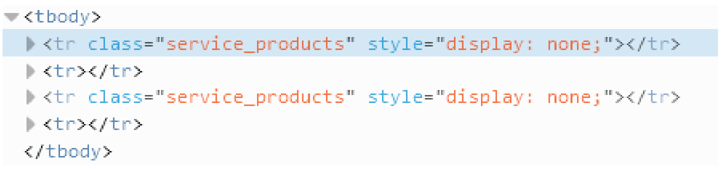
☑ Test automation products only

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 resul\
ts
Thread.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>
  <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.

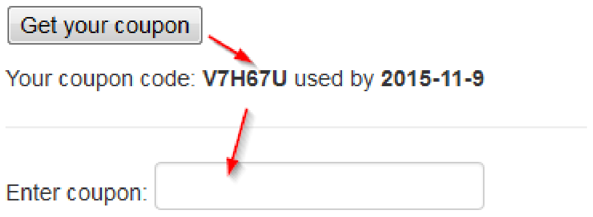
A solution is to extract the visible rows to an array, then we could check them by index.

```
List<WebElement> displayedRows = driver.findElements(By.xpath("//table[@id='\
grid']//tbody/tr[not(contains(@style,'display: none'))]"));
assert(displayedRows.size() == 2);
WebElement firstRowElem = displayedRows.get(0);

String newFirstProductName = firstRowElem.findElement(By.xpath("td[1]")).get\
Text();
assert("BuildWise".equals(newFirstProductName));
firstRowElem.findElement(By.xpath("td/button")).click();
```

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 `<span id='receipt_-number'></span>`. 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.



Your coupon code: **V7H67U** used by **2015-11-9**

Enter coupon:

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.navigate().to(TestHelper.siteUrl() + "/coupon.html");
driver.findElement(By.id("get_coupon_btn")).click();
String couponText = driver.findElement(By.id("details")).getText();

String regex = "Your coupon code:\\s+(\\w+) used by\\s([\\d|-]+)";
Pattern pattern = Pattern.compile(regex);
Matcher matcher = pattern.matcher(couponText);

if (matcher.matches()) {
  String coupon = matcher.group(1);
  String expiryDate = matcher.group(2);
  // System.out.println("coupon = " + coupon);
  assert(coupon.length() == 11);
  driver.findElement(By.name("coupon")).sendKeys(coupon);
} else {
  throw new RuntimeException("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[4] is a helpful tool to let you try out regular expression online.

[4]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.

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[1] (over 430KB in file size), incorrect use of 'text view' will slow your test execution significantly.

```
driver.get(TestHelper.siteUrl()  + "WebDriverStandard.html");
long startTime = System.currentTimeMillis();
String checkText = "language-neutral wire protocol";
assert(driver.findElement(By.tagName("body")).getText().contains(checkText))\
;
double elapsedSec = (System.currentTimeMillis() - startTime) / 1000.0;
System.out.println("Method 1: Search page text took " + elapsedSec + " secon\
ds");
```

[1]http://www.w3.org/TR/webdriver/

```
startTime = System.currentTimeMillis();
assert(driver.getPageSource().contains(checkText));
elapsedSec = (System.currentTimeMillis() - startTime) / 1000.0;
System.out.println("Method 2: Search page HTML took " + elapsedSec + " secon\
ds");
```

Let’s see the difference.

```
Method 1: Search page text took 1.622 seconds
Method 2: Search page HTML took 0.071 seconds
```

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.

```
String checkText = "language-neutral wire protocol";
assert(driver.findElement(By.tagName("body")).getText().contains(checkText));
```

Execution time: **1.619** seconds

```
assert(driver.findElement(By.id("abstract")).getText().contains(checkText));
```

Execution time: **0.032** seconds

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 5 lines of test statements

```
if (refNo.contains("VIP")) { // Special
  assert "".equals(driver.findElement(By.id("notes")).getText());
} else {
  assert "".equals(driver.findElement(By.id("notes")).getText());
}
```

is reduced to one.

```
assert driver.findElement(By.id("notes")).getText().equals( refNo.contains("\
VIP") ? "Please go upstair" : "");
```

Use variable to cache not-changed data

Commonly, I saw people wrote tests like the below to check multiple texts on a page.

```
driver.get(TestHelper.siteUrl()  + "WebDriverStandard.html");
assert(driver.findElement(By.tagName("body")).getText().contains("Firefox"));
assert(driver.findElement(By.tagName("body")).getText().contains("chrome"));
assert(driver.findElement(By.tagName("body")).getText().contains("W3C"));
```

Execution time: **5.27** seconds

The above three test statements are very inefficient, as every test statement calls `driver.findElement(By.`
this can be a quite expensive operation when a web page is large.

Solution: use a variable to store the text (view) of the web page, a very common practice in programming.

```
String thePageText = driver.findElement(By.tagName("body")).getText();
assert(thePageText.contains("Firefox"));
assert(thePageText.contains("chrome"));
assert(thePageText.contains("W3C"));
```

Execution time: **1.61** seconds

As you can see, we get quite constant execution time no matter how many assertions we perform on that page, as long as the page text we are checking is not changing.

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.

```
String longText = new String(new char[5000]).replace('\0', '*');
WebElement textArea = driver.findElement(By.id("comments"));
textArea.sendKeys(longText);
```

Execution time: **3.869** 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.

```
((JavascriptExecutor) driver).executeScript("document.getElementById('commen\
ts').value = arguments[0];", longText);
```

Execution time: **0.021** seconds

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.

```
// static String SITE_URL = "https://physio.clinicwise.net";
static String SITE_URL = "http://demo.poolwise.net";
// static String TARGET_BROWSER = "chrome";
static String TARGET_BROWSER = "firefox";

if (TARGET_BROWSER == "chrome") {
  driver = new ChromeDriver();
} else {
  driver = new FirefoxDriver();
}
driver.get(SITE_URL);
```

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.

```
static String siteRootUrl = "http://sandbox.clinicwise.net"; // default
static WebDriver driver;

// ...

if (System.getenv("BASE_URL") != null && !System.getenv("BASE_URL").isEmpty(\
)) {
     siteRootUrl = System.getenv("BASE_URL");
}

if (browserTypeSetInEnv != null && browserTypeSetInEnv.equals("chrome")) {
    driver = new ChromeDriver();
} else {
    driver = new FirefoxDriver();
}

// ...

driver.navigate.to(siteRootUrl);
```

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.

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.

```
if (System.getenv("BASE_URL") != null && !System.getenv("BASE_URL").isEmpty(\
)) {
  siteRootUrl = System.getenv("BASE_URL");
}
driver.navigate().to(siteRootUrl); // may be dynamically set

if (siteRootUrl.contains("physio.clinicwise.net")) {
  driver.findElement(By.id("username")).sendKeys("natalie");
  driver.findElement(By.id("password")).sendKeys("test");
  driver.findElement(By.id("signin_button")).click();
  assert (driver.getPageSource().contains("Signed in successfully."));
} else if (siteRootUrl.contains("yake.clinicwise.net")) {
  driver.findElement(By.id("username")).sendKeys("tuo");
  driver.findElement(By.id("password")).sendKeys("test");
  driver.findElement(By.id("signin_button")).click();
  assert (driver.getPageSource().contains("□□□□"));
}
```

Though the above test scripts work, it seems lengthy and repetitive.

```
public Boolean isChinese() {
  return siteRootUrl.contains("yake.clinicwise.net");
}

@Test
public void TestTwoLanguagesUsingTernaryOperator() throws Exception {
  driver.navigate().to(siteRootUrl);

  driver.findElement(By.id("username")).sendKeys(isChinese() ? "tuo" : "nata\
lie");
  driver.findElement(By.id("password")).sendKeys("test");
  driver.findElement(By.id("signin_button")).click();
  assert (driver.getPageSource().contains(isChinese() ? "□□□□" : "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("□□")).click();  // chinese
```

Using IDs is much simpler.

```
driver.FindElement(By.id("register_link")).click();
```

This works for all languages.

Multi-language testing with lookups

```
// return the current language used on the site
public String siteLang() {
  if (siteRootUrl.contains("yake.clinicwise.net")) {
    return "chinese";
  } else if (siteRootUrl.contains("sandbox.clinicwise.net")) {
    return "french";
  } else {
    return "english";
  }
}

// in test case
if (siteLang() == "chinese") {
    driver.findElement(By.id("username")).sendKeys("tuo");
} else if (siteLang() == "french") {
    driver.findElement(By.id("username")).sendKeys("dupont");
} else { // default
    driver.findElement(By.id("username")).sendKeys("natalie");
}
driver.findElement(By.id("password")).sendKeys("test");
driver.findElement(By.id("signin_button")).click();
```

If this is going to be used only once, the above is fine. However, these login test steps will be used heavily, which will lead to lengthy and hard to maintain test scripts.

Solution: centralize the logic with lookups.

```
public String userLookup(String username) {
  switch (siteLang()) {
    case "chinese":
      return "tuo";

    case "french":
      return "dupont";

    default:
      return username;
  }
}

// in test case
driver.findElement(By.id("username")).sendKeys(userLookup("natalie"));
```

Astute readers may point out, "You over-simplify the cases, there surely will be more test users." Yes, that's true. I was trying to the simplest way to convey the `lookup`.

```
static Map<String, String> natalieUserDict = new HashMap<String, String>();
static Map<String, String> markUserDict = new HashMap<String, String>();

// ...

@BeforeClass
public static void beforeAll() throws Exception {
  natalieUserDict.put("english", "natalie");
  natalieUserDict.put("chinese", "tuo");
  natalieUserDict.put("french", "dupont");

  markUserDict.put("english", "mark");
  markUserDict.put("chinese", "li");
  markUserDict.put("french", "marc");
}


public String userLookupDict(String username) {
   switch (username) {
     case "natalie":
       return natalieUserDict.get(siteLang());

     case "mark":
       return markUserDict.get(siteLang());

     default:
       return username;
   }
 }
```

```
// In test case
driver.findElement(By.id("username")).sendKeys(userLookupDict("natalie"));
```

In summary, the test user in a chosen language (English in above example) is used as the key to look up for other languages. The equivalent user of "natalie" in French is "dupont".

Some, typically programmers, write the test scripts like the below.

```
public String getAdminUser() {
  // logic goes here
}

driver.findElement(By.id("username")).sendKeys(getAdminUser());
```

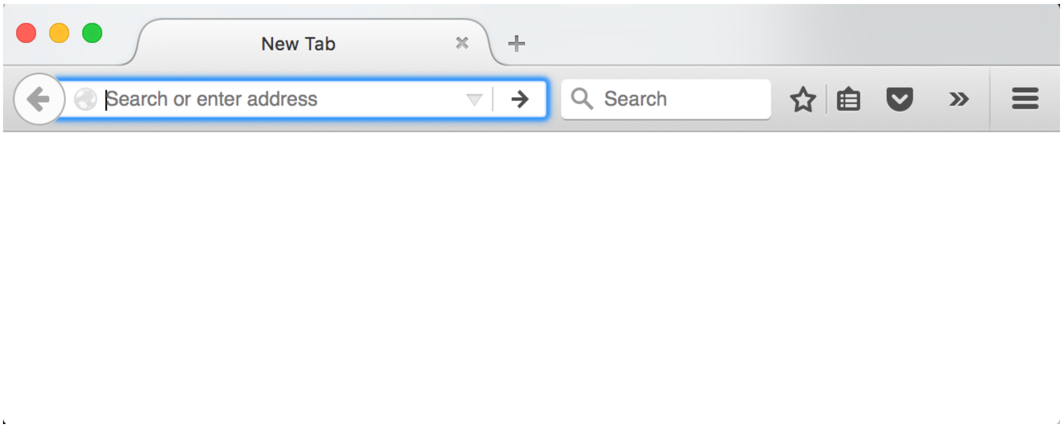
If there are only a handful users, it may be OK. But I often see hard-to-read test statements such as `getRegeisteredUser1()` and `getManager2()`. I cannot say this approach is wrong, I just prefer using 'personas'. But I am against reading test users from external configuration files, which generally I found, hard to maintain.

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.

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.

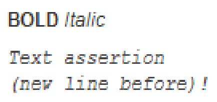
```
driver.findElement(By.id("new_client")).click();
// workaround for chrome v44, make sure the link is clicked
Thread.sleep(1000);
```

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 reliabl\
y with Chrome 44+."
```

Failed to assert copied text in browser

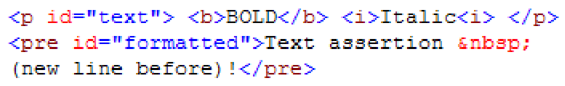
To answer this, let's start with an example. What we see in a browser (Internet Explorer)



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;
(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:

```
// tags in source not in text
assert driver.findElement(By.tagName("body")).getText().contains("BOLD Itali\
c");
assert driver.getPageSource().contains("<b>BOLD</b>  <i>Italic</i>");

// HTML entities in source but shown as space in text
assert driver.findElement(By.tagName("body")).getText().contains("assertion \
 \n(new line before)");
// note 2nd character after assertion is non-breakable space &nbsp;
if (((RemoteWebDriver) driver).getCapabilities().getBrowserName().equals("fi\
refox")) {
   // different behaviour on Firefox (v25)
   assert driver.getPageSource().contains("assertion  \n(new line before)");
} else {
   assert driver.getPageSource().contains("assertion &nbsp;\n(new line befo\
re)");
}
```

The same test works for Chrome, but not for 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.

```
String browserName = ((RemoteWebDriver) driver).getCapabilities().getBrowser\
Name();

if (browserName.equals("chrome")) {
   // chrome specific test statement
} else if (browserName.equals("firefox")) {
   // firefox specific test statement
} else {
   throw new RuntimeException("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.

"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.navigate().to(TestHelper.siteUrl() + "gotchas.html");
new Select(driver.findElement(By.name("vip"))).selectByVisibleText("No");
```

Here is the error returned:

```
UnexpectedTagNameException: Element should have been "select" but was "input"
```

The solution is quite obvious after knowing the cause: change the locator to `new Select(driver.findElement(By.xpath("//select[@name='vip']"))).selectByVisibleText("No");`.

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.

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.

   ```
   WebElement elem = driver.findElement(By.name("submit_action_2"));
   Integer elemPos = elem.getLocation().getY();
     ((JavascriptExecutor) driver).executeScript("window.scroll(0, " + elemPos \
   + ");");
   ```

   Or scroll to the top / bottom of page.

   ```
     ((JavascriptExecutor) driver).executeScript("window.scrollTo(0, document.b\
   ody.scrollHeight);");
   ```

2. A hack, call `sendKeys` to a textfield nearby, if there is one.

23. Selenium Remote Control Server

Selenium Server, formerly known as Selenium Remote Control (RC) Server, allows testers to writes 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.

Selenium Server Installation

Make sure you have Java Runtime installed first. Download Selenium Server *selenium-server-standalone-{VERSION}.jar* from Selenium download page[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-2.40.0.jar
```

Sample output

```
Mar 26, 2014 9:28:31 AM org.openqa.grid.selenium.GridLauncher main
INFO: Launching a standalone server
09:28:31.124 INFO - Java: Oracle Corporation 24.51-b03
09:28:31.125 INFO - OS: Mac OS X 10.9.2 x86_64
09:28:31.142 INFO - v2.40.0, with Core v2.40.0. Built from revision fbe29a9
```

There are two options you can pass to the server: timeout and browserTimeout.

[1]http://www.seleniumhq.org/download/

```
java -jar selenium-server-standalone-2.40.0.jar -timeout=20 -browserTimeout=\
60
```

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 initalization of WebDriver instance to `RemoteWebDriver` like below:

```
static WebDriver driver;

@BeforeClass
public static void beforeAll() throws Exception {
    DesiredCapabilities capabilities = DesiredCapabilities.firefox();
    // if Chrome: DesiredCapabilities.chrome();
    driver = new RemoteWebDriver(capabilities);
}

@Before
public void before() throws Exception {
    driver.get("http://testwisely.com/demo/netbank");
}

 @Test
public void testExplicitWaitsInRemoteBrowser() throws Exception {
    Select select = new Select(driver.findElement(By.name("account")));
    select.selectByVisibleText("Cheque");
    // ...
}
```

```
@Test
public void testImplicitWaitsInRemoteBrowser() throws Exception {
    Select select = new Select(driver.findElement(By.name("account")));
    select.selectByVisibleText("Savings");
    // ...
}

@AfterClass
public static void afterAll() throws Exception {
    driver.quit();
}
```

The test scripts (client) is expected to terminate each browser session properly, calling `driver.quit`.

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-2.40.0.jar -role hub
   ```

2. **Start the nodes**

   A node gets tests from the hub and run them.

   ```
   java -jar selenium-server-standalone-2.40.0.jar -role node  -hub http://loca\
   lhost: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.

```
DesiredCapabilities capabilities = DesiredCapabilities.chrome();
driver = new RemoteWebDriver(new URL("http://127.0.0.1:4444/wd/hub"), capabi\
lities);
// ...
```

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[2] for details.

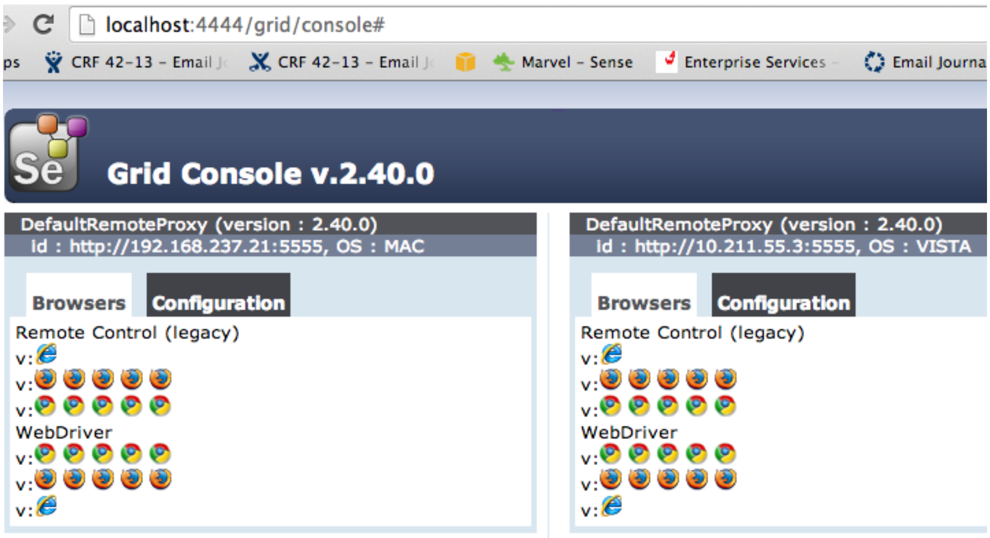
  It is my understanding that just pushing the tests to the hub, and it handles the rest based on the configuration. My experience tells me that it is too good to be true. For example, here is an error I got. While the error message is quite clear: no ChromeDriver installed. But on which node? Shouldn't the hub 'know' about that?

```
[remote server] com.google.common.base.Preconditions(Preconditions.java):177\
:in `checkState': The path to the driver executable must be set by the webdr\
iver.chrome.driver system property; for more information, see http://code.go\
ogle.com/p/selenium/wiki/ChromeDriver. The latest version can be downlo
from http://chromedriver.storage.googleapis.com/index.html (java.lang.I
lStateException) (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').

[2]https://code.google.com/p/selenium/wiki/Grid2


localhost:4444/grid/console#
Grid Console v.2.40.0
DefaultRemoteProxy (version : 2.40.0)
id : http://192.168.237.21:5555, OS : MAC
Browsers
Configuration
Remote Control (legacy)
WebDriver
DefaultRemoteProxy (version : 2.40.0)
id : http://10.211.55.3:5555, OS : VISTA
Browsers
Configuration
Remote Control (legacy)
WebDriver

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 anther 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 prefect 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

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

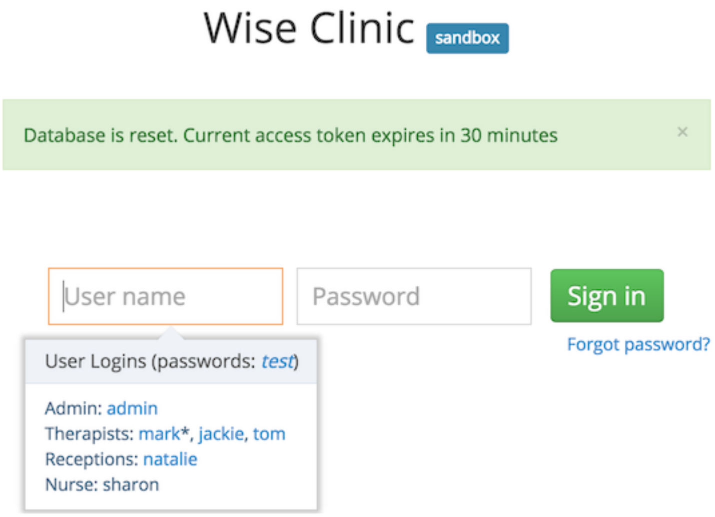
Like any other skills, you will get better at it by practising more.

- **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: ClinicWise sandbox site[4]. ClinicWise is a modern web application using popular web technologies such as AJAX and Bootstrap. I have written 429 Selenium WebDriver tests for ClinicWise. Execution of all tests takes more than 3 hours on a single machine. If you like, you can certainly practise writing tests against ClinicWise sandbox.

  ClinicWise is also a show case of web applications designed for testing, which means it is easier to write automated tests against it. Our every Selenium test starts with calling a database reset: visit *http:///sandbox.clinicwise.net/reset*, which will reset the database to a seeded state.

[4]http:///sandbox.clinicwise.net


Wise Clinic
sandbox
Database is reset. Current access token expires in 30 minutes
×
User name
Password
Sign in
Forgot password?
User Logins (passwords: *test*)
Admin: admin
Therapists: mark*, jackie, tom
Receptions: natalie
Nurse: sharon

- **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[5].

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*[6].

[5]https://leanpub.com/learn-ruby-programming-by-examples-en
[6]https://leanpub.com/practical-web-test-automation

- **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. *Practical Web Test Automation* has one chapter on this.

Best wishes for your test automation!

Resources

Books

- **Practical Web Test Automation**[7] 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:
  - Developing easy to read and maintain Watir/Selenium tests using next-generation functional testing tool
  - Page object model
  - Functional Testing Refactorings
  - Cross-browser testing against IE, Firefox and Chrome
  - Setting up continuous testing server to manage execution of a large number of automated UI tests
  - Requirement traceability matrix
  - Strategies on team collaboration and test automation adoption in projects and organizations
- **Selenium WebDriver Recipes in C#**[8] by Zhimin Zhan

  Selenium WebDriver recipe tests in C#, another popular language that is quite similar to Java.
- **Selenium WebDriver Recipes in Ruby**[9] 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**[10] by Zhimin Zhan

  Selenium WebDriver recipes in Python, a popular script language that is similar to Ruby.

[7]https://leanpub.com/practical-web-test-automation
[8]https://leanpub.com/selenium-recipes-in-csharp
[9]https://leanpub.com/selenium-recipes-in-ruby
[10]https://leanpub.com/selenium-recipes-in-python

Web Sites

- **Selenium Java API** http://selenium.googlecode.com/git/docs/api/java/index.html[11]
- **Selenium Home** (http://seleniumhq.org[12])

Tools

- **NetBeans IDE** (https://netbeans.org/downloads[13])

  Free Java IDE from Sun (now Oracle).
- **BuildWise** (http://testwisely.com/buildwise[14])

  AgileWay's free and open-source continuous build server, purposely designed for running automated UI tests with quick feedback.

[11]http://selenium.googlecode.com/git/docs/api/java/index.html
[12]http://seleniumhq.org
[13]https://netbeans.org/downloads
[14]http://testwisely.com/buildwise