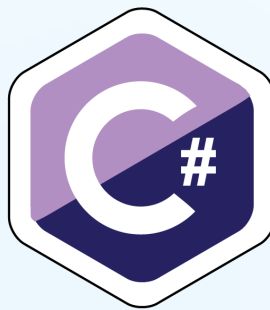


# Selenium WebDriver Recipes in C#

The Problem Solving Guide to Selenium WebDriver



Zhimin Zhan

# Selenium WebDriver Recipes in C#

The problem solving guide to Selenium WebDriver in C#

Zhimin Zhan

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

This version was published on 2015-07-16



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

©2014 - 2015 Zhimin Zhan

# Contents

<b>1. Preface</b>	<b>1</b>
1.1 Who should read this book	2
1.2 How to read this book	2
1.3 Recipe test scripts	2
1.4 Send me feedback	2
<b>2. Introduction</b>	<b>3</b>
2.1 Selenium	3
2.2 Selenium language bindings	3
2.3 Cross browser testing	11
2.4 Visual Studio Unit Testing Framework	14
2.5 Alternative framework NUnit	16
2.6 Run recipe scripts	16
<b>3. Locating web elements</b>	<b>21</b>
3.1 Start browser	21
3.2 Find element by ID	21
3.3 Find element by Name	22
3.4 Find element by Link Text	22
3.5 Find element by Partial Link Text	22
3.6 Find element by XPath	23
3.7 Find element by Tag Name	24
3.8 Find element by Class	24
3.9 Find element by CSS Selector	25
3.10 Chain FindElement to find child elements	25
3.11 Find multiple elements	25
<b>4. Hyperlink</b>	<b>27</b>
4.1 Click a link by text	27

## CONTENTS

4.2	Click a link by ID . . . . .	27
4.3	Click a link by partial text . . . . .	28
4.4	Click a link by XPath . . . . .	28
4.5	Click Nth link with exact same label . . . . .	29
4.6	Click Nth link by CSS Selector . . . . .	29
4.7	Verify a link present or not? . . . . .	30
4.8	Getting link data attributes . . . . .	30
4.9	Test links open a new browser window . . . . .	30

# 1. 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](#)<sup>1</sup>. 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. If you have read one of my other books: *Practical Web Test Automation*<sup>2</sup>, you probably know my style: **practical**. I will let the test scripts do most of the 'talking'. These recipe test scripts are 'live', as I have created the target test site and included offline test web pages. With both, you can:

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

---

<sup>1</sup><http://www.wired.com/business/2013/04/linkedin-software-revolution/>

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

## 1.1 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 C# coding skill is required.

## 1.2 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.

## 1.3 Recipe test scripts

To help readers to learn more effectively, this book has a [dedicated site](#)<sup>3</sup> 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.

## 1.4 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*

December 2014

---

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

## 2. 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).

### 2.1 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.

### 2.2 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 C# 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 C# programmers develop C# code in Visual Studio (Integrated Development Environment). I will use Visual Studio Express 2013 as the IDE of choice for this book, as it is free.

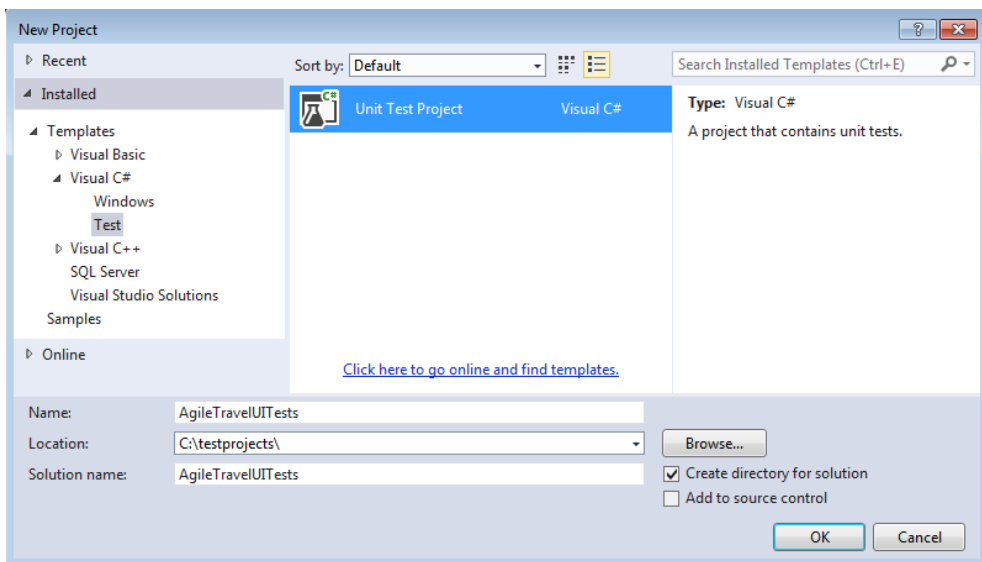
### Prerequisite:

- Download and install Visual Studio 2013.
- Your target browser is installed, such as Chrome or Firefox.

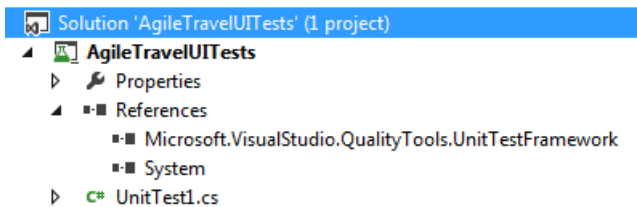
## Set up Visual Studio Solution

1. Create a new project in Visual Studio

Select template 'Templates' → 'Visual C#' → 'Test' → 'Unit Test Project'.



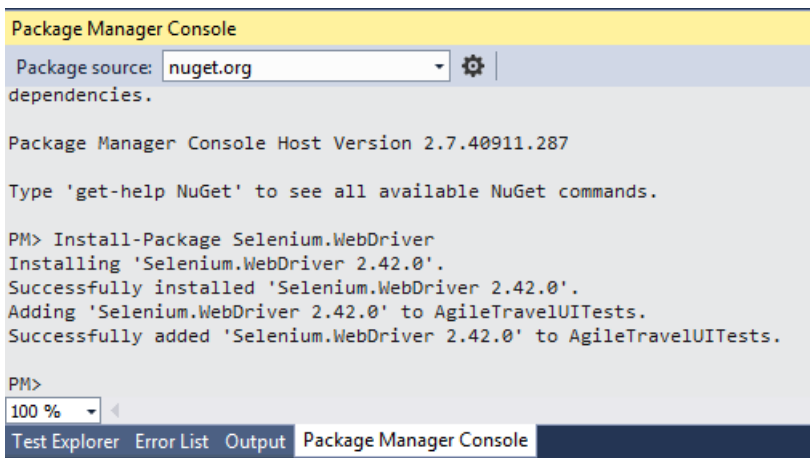
You shall see the project skeleton created.



2. Add Selenium WebDriver package to the project

Run the following command in 'Package Manager Console' (selecting menu 'Tools' → 'Library Package Manager' → 'Package Manager Console').

```
PM> Install-Package Selenium.WebDriver
```



```
Package Manager Console
Package source: nuget.org
dependencies.

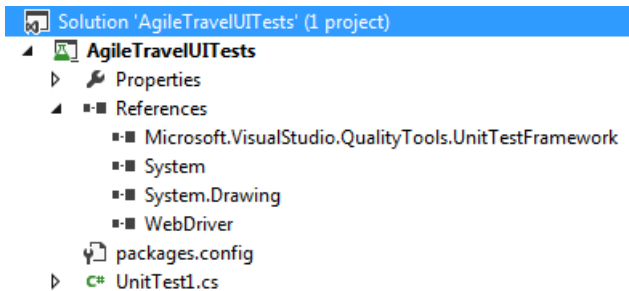
Package Manager Console Host Version 2.7.40911.287

Type 'get-help NuGet' to see all available NuGet commands.

PM> Install-Package Selenium.WebDriver
Installing 'Selenium.WebDriver 2.42.0'.
Successfully installed 'Selenium.WebDriver 2.42.0'.
Adding 'Selenium.WebDriver 2.42.0' to AgileTravelUITests.
Successfully added 'Selenium.WebDriver 2.42.0' to AgileTravelUITests.

PM>
```

Here is what looks like in Visual Studio's Solution Explorer after Selenium WebDriver was installed.



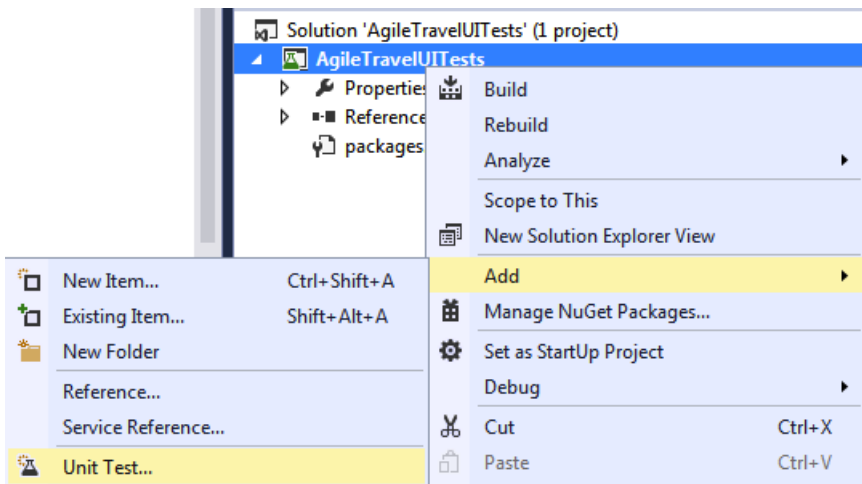
Also install “Selenium.Support”, which includes helper .NET classes for the Selenium WebDriver API.

```
PM> Install-Package Selenium.Support
```

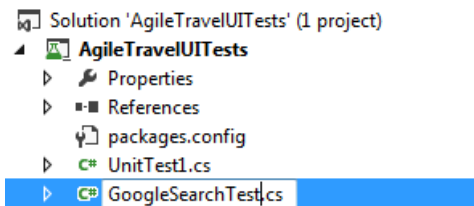
## Create a test and run it

1. Add a new C# class (a test)

Essentially a Selenium test in C# is a C# Class. Right mouse click the project name (in Solution Explorer) select ‘Add’ → ‘Unit Test’.



A new unit test file is created (named `UnitTest1.cs` or `UnitTest2.cs` if `UnitTest1.cs` already exists). Rename it (press F2 key and enter a name in Camel Case, such as `SayHelloWorld`)



## 2. Create a test case

For simplicity, paste a google search test case script into the editor.

```

[TestClass]
public class GoogleSearchTest
{
    [TestMethod]
    public void TestSearch()
    {
        IWebDriver driver = new FirefoxDriver();

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

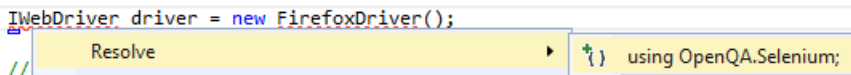
        // Find the text input element by its name
        IWebElement element = driver.FindElement(By.Name("q"));

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

        // Now submit the form.
        element.Submit();
    }
}

```

The several red underlines indicate compiling errors. In this case, the Selenium classes were not loaded. To fix, right mouse click these red underlines and select 'Resolve'.



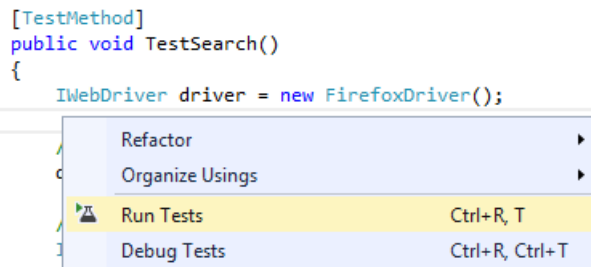
```

IWebDriver driver = new FirefoxDriver();
//

```

### 3. Run the test

Right click the editor and select 'Run Tests'.



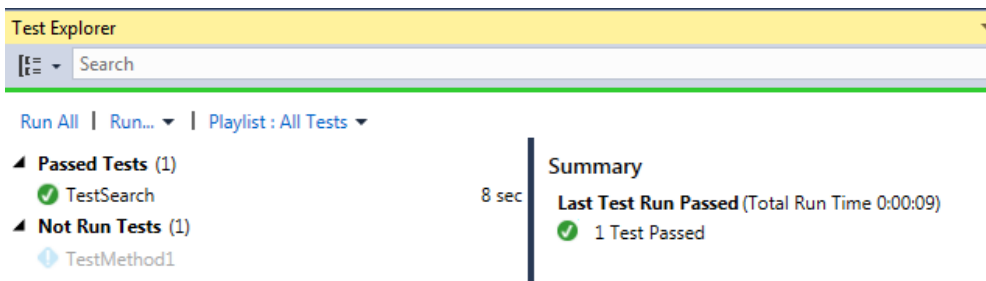
```

[TestMethod]
public void TestSearch()
{
    IWebDriver driver = new FirefoxDriver();
}

```

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

Click 'Test Explorer' tab to see test execution result.



## 2.3 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](#)<sup>1</sup>, 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 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 (not too outdated version) installed, you are ready to go. The test script below will open a web site in a new Firefox window.

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

### Chrome

To run Selenium tests in Google Chrome, besides the Chrome browser itself, [ChromeDriver](#)<sup>2</sup> needs to be installed.

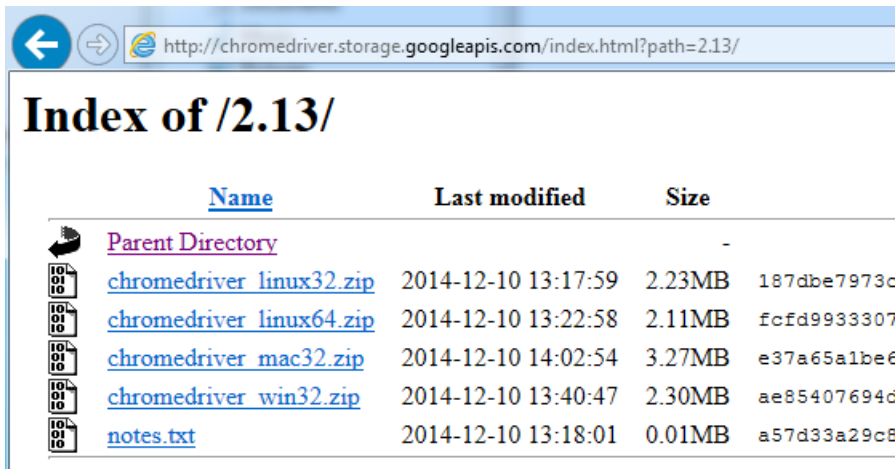
Installing ChromeDriver is easy: go to <http://chromedriver.storage.googleapis.com/index.html><sup>3</sup>

---

<sup>1</sup>[http://en.wikipedia.org/wiki/Usage\\_share\\_of\\_web\\_browsers](http://en.wikipedia.org/wiki/Usage_share_of_web_browsers)

<sup>2</sup><http://chromedriver.storage.googleapis.com>

<sup>3</sup><http://chromedriver.storage.googleapis.com/index.html>



download the one for your target platform, unzip it and put chromedriver executable in your PATH. To verify the installation, open a command window, execute command *chromedriver*. You shall see:

```
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.

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

## Internet Explorer

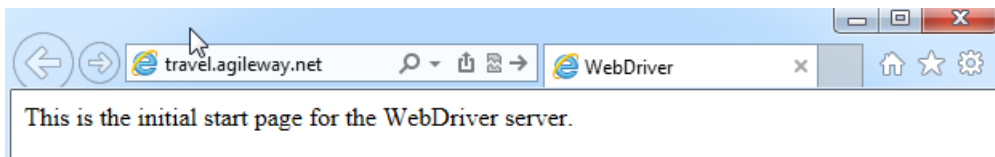
Selenium requires IEDriverServer to drive IE browser. Its installation process is very similar to *chromedriver*. IEDriverServer is available at <http://www.seleniumhq.org/download/><sup>4</sup>. Choose the right one based on your windows version (32 or 64 bit).

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

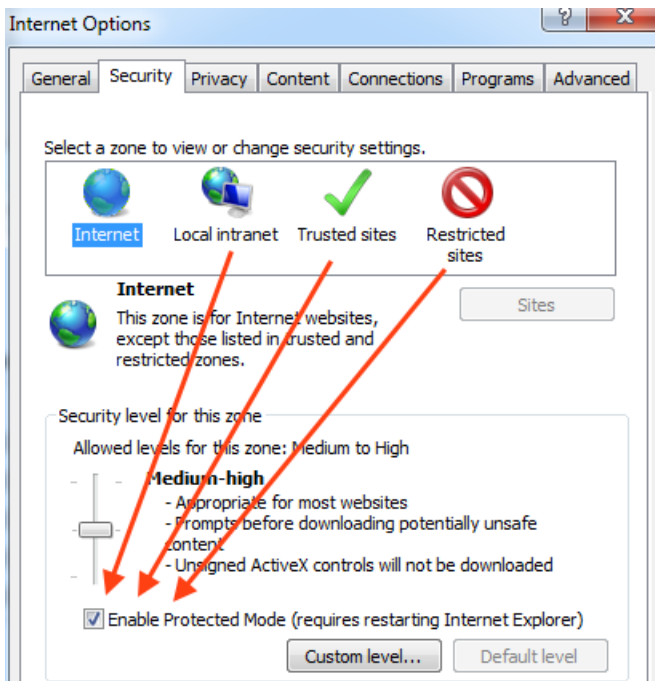
When a tests starts to execute in IE, before navigating the target test site, you will see this first:

<sup>4</sup><http://www.seleniumhq.org/download/>





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).



Further configuration is required for IE10 and IE11, see [IE and IEDriverServer Runtime Configuration](#)<sup>5</sup> for details.

```
using OpenQA.Selenium.IE;
```

```
//...
```

```
IWebDriver driver = new InternetExplorerDriver();
```

<sup>5</sup>[https://code.google.com/p/selenium/wiki/InternetExplorerDriver#Required\\_Configuration](https://code.google.com/p/selenium/wiki/InternetExplorerDriver#Required_Configuration)

## 2.4 Visual Studio Unit Testing Framework

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). Here is an example using Visual Studio Unit Testing Framework.

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using OpenQA.Selenium.Firefox;
using OpenQA.Selenium.IE;
using OpenQA.Selenium.Chrome;
using System.Collections.ObjectModel;

[TestClass]
public class GoogleSearchDifferentBrowsersTest {

    [TestMethod]
    public void TestInIE() {
        IWebDriver driver = new InternetExplorerDriver();
        driver.Navigate().GoToUrl("http://testwisely.com/demo");
        System.Threading.Thread.Sleep(1000);
        driver.Quit();
    }

    [TestMethod]
    public void TestInFirefox() {
        IWebDriver driver = new FirefoxDriver();
        driver.Navigate().GoToUrl("http://testwisely.com/demo");
        System.Threading.Thread.Sleep(1000);
        driver.Quit();
    }

    [TestMethod]
    public void TestInChrome() {
        IWebDriver driver = new ChromeDriver();
        driver.Navigate().GoToUrl("http://testwisely.com/demo");
    }
}
```

```
        System.Threading.Thread.Sleep(1000);
        driver.Quit();
    }
}
```

@TestMethod annotates a test case below, in a format of testCamelCase() or TestCapital-Case. You will find more about VS Unit Test Framework from [its home page](#)<sup>6</sup>. 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 it.

## Visual Studio Unit Testing Framework fixtures

If you worked with xUnit before, you must know setUp() and tearDown() fixtures, used run before or after every test. In VS Unit Test Framework, by using annotations ([ClassInitialize], [TestInitialize], [TestCleanup], [ClassCleanup]), you can choose the name for fixtures. Here are mine:

```
[ClassInitialize]
public static void BeforeAll() {
    // run before all test cases
}
```

```
[TestInitialize]
public void Before() {
    // run before each test case
}
```

```
[TestMethod]
public void TestCase1() {
    // one test case
}
```

```
[TestMethod]
public void TestCase2() {
```

---

<sup>6</sup><http://msdn.microsoft.com/en-us/library/ms243147>

```
// another test case
}

[TestCleanup]
public void After() {
    // run after each test case
}

[ClassCleanup]
public static void AfterAll() {
    // run after all test cases, typically, close browser
}
```

Please note that `ClassCleanup` does not guarantee all tests from one test class have been executed before a new test class is initialized. What does this mean to your execution? If you run one test case or all test cases in one test class, `[ClassCleanup]` is invoked as you expected. However, when you run several test classes in one go, the time of invocation of `[ClassCleanup]` is nondeterministic (which lead many browser windows open). In my opinion, it sucks. Read this MSDN blog post for explanation: “[ClassCleanup May Run Later Than You Think](http://blogs.msdn.com/b/ploeh/archive/2007/01/06/classcleanupmayrunlaterthanyouthink.aspx)”<sup>7</sup>.

## 2.5 Alternative framework NUnit

[NUnit](http://nunit.org/)<sup>8</sup>, inspired by JUnit, is an open source unit testing framework for Microsoft .NET. If you have used JUnit before, you will find similarities in NUnit. Personally, I like NUnit more (comparing to MS Unit Test Framework). In particular, it’s more flexible to execute tests from command line with JUnit style test reports (which can be easily integrated with CI Servers).

However, you will need to spend some effort getting NUnit to work with Visual Studio.

## 2.6 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 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 live sites requires Internet connection.

---

<sup>7</sup><http://blogs.msdn.com/b/ploeh/archive/2007/01/06/classcleanupmayrunlaterthanyouthink.aspx>

<sup>8</sup><http://nunit.org/>

## Run tests in Visual Studio

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, the most effective way to run all tests is done by a Continuous Integration process)

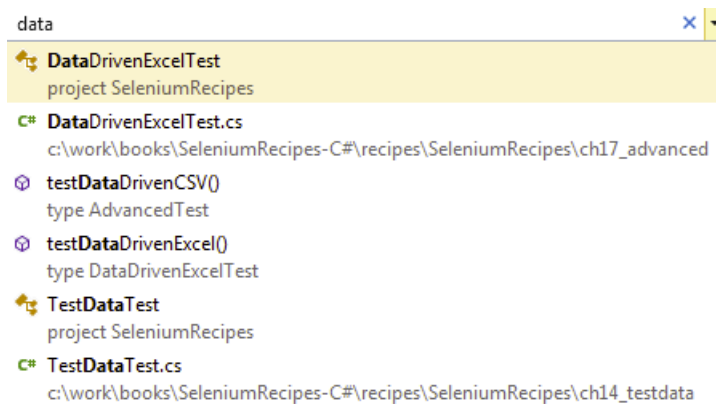
### Find the test case

You can locate the recipe either by following the chapter or searching by name. There are over 100 test cases in one test project. Being able to quickly navigate to the test case is important when you have developed a large number of test cases.

Visual Studio has the “Navigate To” command, its default keyboard shortcut is **Ctrl + ’, ’**.



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

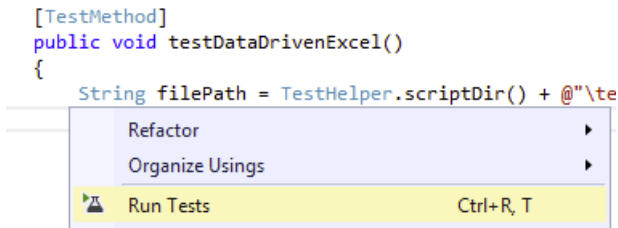


### Run individual test case

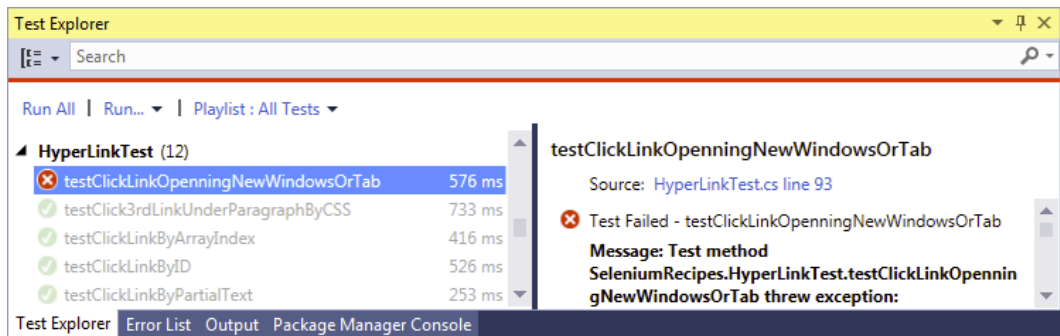
Move caret to a line within a test case, in

```
[TestMethod]
public void TestCase2() {
    // one test case
}
```

Right mouse click and select “Run Tests” to run this case.

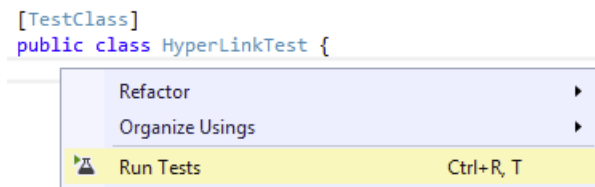


The below is a screenshot of execution panel when one test case failed,

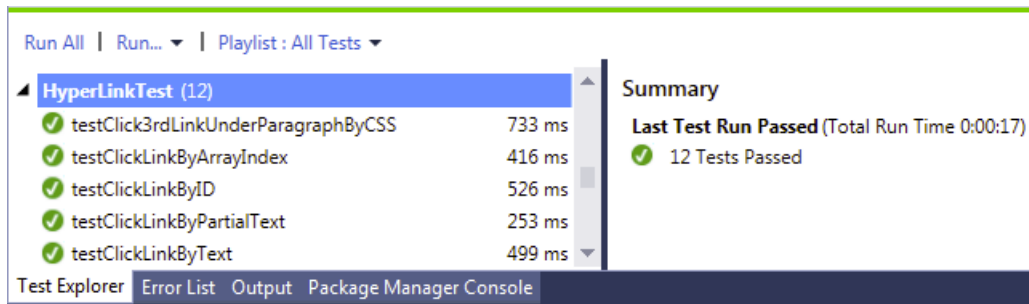


## Run all test cases in a test script file

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’.

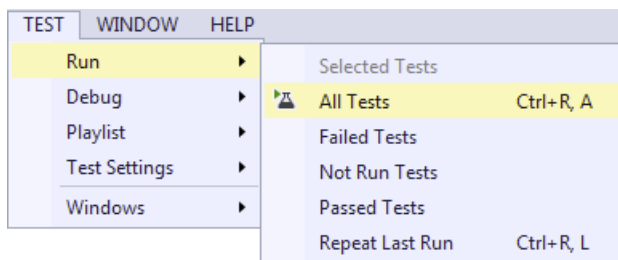


The below is a screenshot of the execution panel when all test cases in a test script file passed,

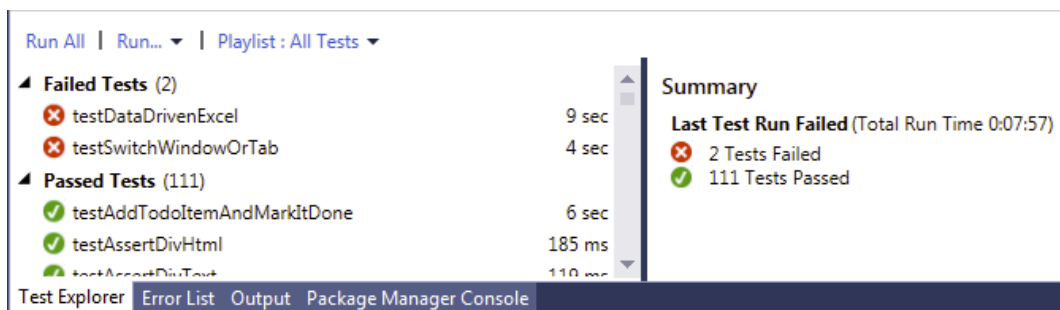


## Run all tests

You can also run all test cases in a Visual Studio project. Firstly, set the main project by selecting menu 'Tools' → 'Run' → 'All Tests'.



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



## 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 C# class, you need to be compile it first (Within IDE, IDEs do it for you automatically).

1. Open a Visual Studio command prompt.

Click “Start” -> “All Programs” -> “Visual Studio 2013” -> “Visual Studio Tools”, and choose “Developer Command Prompt for VS2013”.

By default, the Visual Studio command prompt opens to the following folder:

```
<drive letter>:\Program Files (x86)\Microsoft Visual Studio 12.0>
```

2. Change directory to your solution folder

```
cd C:\work\books\SeleniumRecipes-C#\recipes\SeleniumRecipes\bin\Debug
```

3. Run VSTest.Console program

```
vstest.console SeleniumRecipes.dll
```

## Example Output

```
Total tests: 113. Passed: 110. Failed: 3. Skipped: 0.  
Test Run Failed.  
Test execution time: 8.8321 Minutes
```



## 3. 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	<code>FindElement(By.Id("user"))</code>
Name	<code>FindElement(By.Name("username"))</code>
Link Text	<code>FindElement(By.LinkText("Login"))</code>
Partial Link Text	<code>FindElement(By.PartialLinkText("Next"))</code>
XPath	<code>FindElement(By.XPath("//div[@id='login']/input"))</code>
Tag Name	<code>FindElement(By.TagName("body"))</code>
Class Name	<code>FindElement(By.ClassName("table"))</code>
CSS	<code>FindElement(By.CssSelector, "#login &gt; input[type='text']")</code>

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

### 3.1 Start browser

Testing websites starts with a browser.

```
static WebDriver driver = new FirefoxDriver();
driver.Navigate().GoToUrl("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();
```

### 3.2 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<sup>1</sup>](http://www.w3.org/TR/WCAG20-TECHS/H93.html), the IDs should be unique and identified in web controls. In comparison

---

<sup>1</sup><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();    // Button
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
```

### 3.3 Find element by Name

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

```
driver.FindElement(By.Name("comment")).SendKeys("Selenium Cool");
```

### 3.4 Find element by Link Text

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

```
driver.FindElement(By.LinkText("Cancel")).Click();
```

### 3.5 Find element by Partial Link Text

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

```
// will click the "Cancel" link
driver.FindElement(By.PartialLinkText("ance")).Click();
```

## 3.6 Find element by XPath

XPath, the XML Path Language, is a query language for selecting nodes from an XML document. When a browser renders a web page, it parses it into a DOM tree or similar. XPath can be used to refer a certain node in the DOM tree. If this sounds 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*

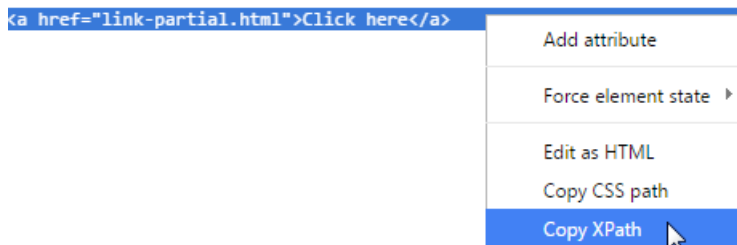
```
driver.FindElement(By.XPath("//*[@id='div2']/input[@type='checkbox']")).Click();
```

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



### Avoid using copied XPath from Browser's Developer Tool

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



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

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

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

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

### 3.7 Find element by Tag Name

There are a limited set of tag names in HTML. In other words, many elements share the same tag names on a web page. We normally don't use the `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")).Text;
```

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.

### 3.8 Find element by Class

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

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

You may use any one of them.

```
driver.FindElement(By.ClassName("btn-primary")).Click(); // Submit button \
```

```
driver.FindElement(By.ClassName("btn")).Click(); // Cancel link
```

```
// the below will return error "Compound class names not permitted"
// driver.FindElement(By.ClassName("btn btn-default 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();
System.Threading.Thread.Sleep(500);
driver.FindElement(By.ClassName("editable-textarea")).SendKeys("inline notes\
");
System.Threading.Thread.Sleep(500);
driver.FindElement(By.ClassName("editable-submit")).Click();
```

## 3.9 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.

## 3.10 Chain FindElement to find child elements

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

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

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

```
driver.FindElement(By.Id("div2")).FindElement(By.Name("same")).Click();
```

## 3.11 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.

```
ReadOnlyCollection<IWebElement> checkbox_elems = driver.FindElements(By.XPath\  
h("//div[@id='container']//input[@type='checkbox']"));  
System.Console.WriteLine(checkbox_elems); // => 2  
checkbox_elems[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.

## 4. 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>
```

### 4.1 Click a link by text

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

```
driver.FindElement(By.LinkText("Recommend Selenium")).Click();
```

### 4.2 Click a link by ID

```
driver.FindElement(By.Id("recommend_selenium_link")).Click();
```

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

```
if (isItalian()) {  
    driver.FindElement(By.LinkText("Accedi")).Click();  
} else if (isChinese()) { // a helper function determines the locale  
    driver.FindElement(By.LinkText, "登录").Click();  
} else {  
    driver.FindElement(By.LinkText("Sign in")).Click();  
}
```

### 4.3 Click a link by partial text

```
driver.FindElement(By.PartialLinkText("Recommend Seleni")).Click();
```

### 4.4 Click a link by XPath

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

```
driver.FindElement(By.XPath("//p/a[text()='Recommend Selenium']")).Click();
```

You 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();

```

## 4.5 Click Nth link with exact same label

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

The web page below contains 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.IsTrue( driver.FindElements(By.LinkText("Show Answer")).Count == 2);
ReadOnlyCollection<IWebElement> links = driver.FindElements(By.LinkText("Sho\\
w Answer"));
links[1].Click(); // click the second one
Assert.IsTrue(driver.PageSource.Contains("second link page")); // second link

```

`FindElements` return a collection (also called array by some) of web controls matching the criteria in appearing order. Selenium (in fact C#) uses 0-based indexing, i.e., the first one is 0.

## 4.6 Click Nth link by CSS Selector

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

```
// Click the 3rd link directly in <p> tag
driver.FindElement(By.CssSelector("p > a:nth-child(3)")).Click();
```

However, generally speaking, the use of stylesheet is more prone to changes.

## 4.7 Verify a link present or not?

```
Assert.IsTrue(driver.FindElement(By.LinkText("Recommend Selenium")).Displaye\
d);
Assert.IsTrue(driver.FindElement(By.Id("recommend_selenium_link")).Displaye\
);
```

## 4.8 Getting link data attributes

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

```
Assert.AreEqual(TestHelper.SiteUrl(), driver.FindElement(By.LinkText("Recom\
mend Selenium")).GetAttribute("href"));
Assert.AreEqual("recommend_selenium_link", driver.FindElement(By.LinkText("R\
ecommend Selenium")).GetAttribute("id"));
Assert.AreEqual("Recommend Selenium", driver.FindElement(By.Id("recommend_se\
lenium_link")).Text);
Assert.AreEqual("a", driver.FindElement(By.Id("recommend_selenium_link")).Ta\
gName);
```

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

```
Assert.AreEqual("font-size: 14px;", driver.FindElement(By.Id("recommend_sele\
nium_link")).GetAttribute("style"));
// Please note using attribute_value("style") won't work
Assert.AreEqual("123", driver.FindElement(By.Id("recommend_selenium_link")).\
GetAttribute("data-id"));
```

## 4.9 Test links open a new browser window

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

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

While we could use `switchTo()` method (see chapter 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.Url;  
String newWindowUrl = driver.FindElement(By.LinkText("Open new window")).Get\  
Attribute("href");  
driver.Navigate().GoToUrl(newWindowUrl);  
driver.FindElement(By.Name("name")).SendKeys("sometext");  
driver.Navigate().GoToUrl(currentUrl); // back
```

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