



INSTANT

Short | Fast | Focused

Selenium Testing Tools Starter

A short, fast, and focused guide to Selenium Testing tools that delivers immediate results

Unmesh Gundecha

[PACKT]
PUBLISHING

Instant Selenium Testing Tools Starter

A short, fast, and focused guide to Selenium Testing tools that delivers immediate results

Unmesh Gundecha



BIRMINGHAM - MUMBAI

Instant Selenium Testing Tools Starter

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: April 2013

Production Reference: 1170413

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78216-514-9

www.packtpub.com

Credits

Author

Unmesh Gundecha

Project Coordinator

Joel Goveya

Reviewers

Vatsala Dorairajan

Murat Gocmen

Proofreader

Aaron Nash

Acquisition Editor

Usha Iyer

Graphics

Valentina D'silva

Commissioning Editor

Neha Nagwekar

Production Coordinator

Melwyn D'sa

Technical Editor

Jalasha D'costa

Cover Work

Melwyn D'sa

Cover Image

Conidon Miranda

About the Author

Unmesh Gundecha has a Master's Degree in Software Engineering and around 10 years of experience in software development and testing. Unmesh has architected the functional test automation projects using industry standards, in-house and custom test automation frameworks, along with leading commercial and open source test automation tools. Presently, he is working as a Test Architect with a multinational company in Pune, India.

He is also the author of *Selenium Testing Tools Cookbook*, published by *Packt Publishing* in November 2012.

I would like to thank my family who are always supportive in everything I do and especially my two lovely kids, Ira and Arav. I would also like to thank the Packt team and reviewers for giving the perfect shape to this book.

Finally, big thanks to Selenium Development and User Community for building this wonderful tool.

About the Reviewers

Vatsala Dorairajan is a budding software technologist. In her 4 years of work experience spread across three startup companies, she has worked with 'Idea'smiths, building 'on paper'/'in-concept' ideas into working prototypes evolving into products. Her technical experience so far has been in Java, Flex, Python, and PHP. She is passionate about making classroom education an absolutely fun experience and hopes to be an educationist one day.

Murat Gocmen is a QA Automation Engineer at IBM and has an overall experience of 5 years in the software industry. He has completed his Master's in Computer Engineering from Air Force Institute of Technology (AFIT) and has studied Bachelor of Science in Computer Science and Engineering. He has worked in various sectors such as web commerce, retail, and apartment-rental.

www.packtpub.com

Support files, eBooks, discount offers, and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt Publishing offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.packtpub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.packtpub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt Publishing books and eBooks.

www.packtlib.packtpub.com

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- ◆ Fully searchable across every book published by Packt
- ◆ Copy and paste, print, and bookmark content
- ◆ On demand and accessible via web browser

Free Access for Packt Publishing account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.



Table of Contents

Instant Selenium Testing Tools Starter	1
So, what is Selenium?	3
The basic features of Selenium	3
What kind of things can you do with it?	5
How can you use this technology within your existing project?	5
Installation	6
Step 1 – What do I need?	6
Step 2 – Downloading Selenium IDE	6
Step 3 – Installing Selenium IDE	7
And that's it!	9
Quick start	10
Step 1 – Recording and adding commands in a test	10
Step 2 – Saving the recorded test	12
Step 3 – Saving the test suite	13
Step 4 – Running the recorded test	13
Step 5 – Exporting a recorded test to Selenium WebDriver	14
Top 5 features you'll want to know about	22
1 – Running tests on various browsers	22
2 – Locating elements	26
3 – Working with HTML elements	29
4 – Synchronizing steps	30
Selenium IDE	30
Selenium WebDriver	31
5 – The Page Object pattern	33

Table of Contents

People and places you should get to know	36
Official sites	36
Articles and tutorials	36
Community	36
Blogs	37
Twitter	37

Instant Selenium Testing Tools Starter

Welcome to *Instant Selenium Testing Tools Starter*. This book has been especially created to provide you with all the information that you need to get up to speed with Selenium IDE and Selenium WebDriver. You will learn the basics of Selenium, get started with installing Selenium, creating a test suite and tests cases, and then running these tests on your web application, using WebDriver, and some tips and tricks for using Selenium.

This book contains the following sections:

So what is Selenium? helps you find out what Selenium actually is, what you can do with it, and why it's so great.

Installation teaches you how to download and install Selenium with the minimum fuss and then set it up so that you can use it as soon as possible.

Quick start teaches you how to record a test, save a test case, enhance a test by adding commands, and run a test with Selenium IDE. This section will also get you started on programming with Selenium WebDriver. Here you will learn how to perform some core tasks in Selenium WebDriver such as creating an instance of a desired browser. It explains interacting with page elements using WebElement, adding verification points, and writing user defined methods.

Top 5 features you'll want to know about will introduce you to various features that Selenium provides in order to automate your web applications. This section will introduce you to some of the key features of Selenium such as running tests on various browsers, locating elements, working with HTML elements, synchronizing steps, and the Page Object pattern.

People and places you should get to know provides you with many useful links to the project page and forums, as well as a number of helpful articles, tutorials, blogs, and the Twitter feeds of Selenium super-contributors, as every open source project is centered around a community.

So, what is Selenium?

In this section, you will get to know a bit about Selenium; its basic features, what you can do with it, and how you can put it to work with automating tasks in a browser window, or build automated tests to validate your web application.

The basic features of Selenium

Selenium is a browser automation framework. It provides a number of tools and APIs for automating user interaction on pure HTML and JavaScript applications in browsers such as IE, Firefox, Google Chrome, Safari, and many more. However, Selenium does not support **Rich Internet Application (RIA)** technologies such as Silverlight, Flex/Flash, and JavaFx out of the box.

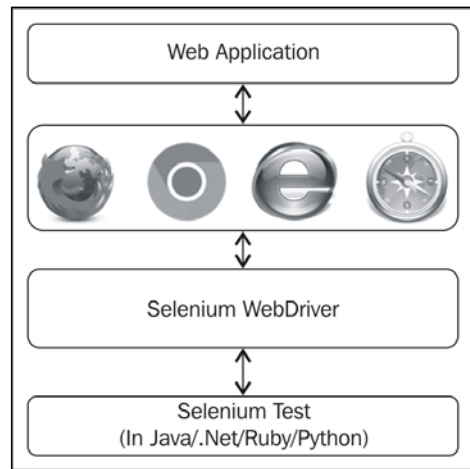
Selenium offers the following set of tools for automating interaction with browsers:



- ◆ **Selenium IDE:** This is a Firefox add-on for recording and playing back Selenium scripts with Firefox. It provides a GUI for recording user actions using Firefox. It's a great tool to start learning and using Selenium, but it can only be used with Firefox while other browsers are not supported.

However, you can convert the recorded scripts into various programming languages supported by Selenium WebDriver and run these scripts on browsers other than Firefox.

- ◆ **Selenium WebDriver:** This is a programming interface for developing more advanced Selenium scripts using different programming languages. You can also run tests on multiple browsers supported by Selenium. The following figure provides a high-level architecture of Selenium WebDriver:



Selenium WebDriver supports browsers including Mozilla Firefox, Google Chrome, Microsoft Internet Explorer, Safari, and Opera.

It supports writing scripts with various programming languages including Java, .NET Languages (C#, VB.NET), Python, Ruby, PHP, and JavaScript.

- ◆ **Selenium Standalone Server:** This allows remote and distributed execution of Selenium scripts. You can also use the Grid feature of a standalone server to run tests in parallel and run tests on mobile platforms such as Android or Apple iOS for iPhone and iPad.

What kind of things can you do with it?

Selenium is widely used for automated testing of web applications; however, its usage is not limited to testing. Selenium mimics user actions such as entering text into a text field, clicking on buttons or links, selecting an option from a drop-down list, and many more, in a browser window as if a human user is interacting with the application. Selenium is also used for screen scraping and automating repetitive tasks in web applications.

How can you use this technology within your existing project

You can use Selenium for functional/acceptance testing of your web applications. You can create automated regression tests using Selenium and run them whenever you need to test a new build of your application.

You can use Selenium to automate repetitive tasks such as data entry, filling out forms, check status, or perform complex navigation steps while manual testing.

Installation

In three easy steps, you can install Selenium IDE and get it set up on your system.

Step 1 – What do I need?

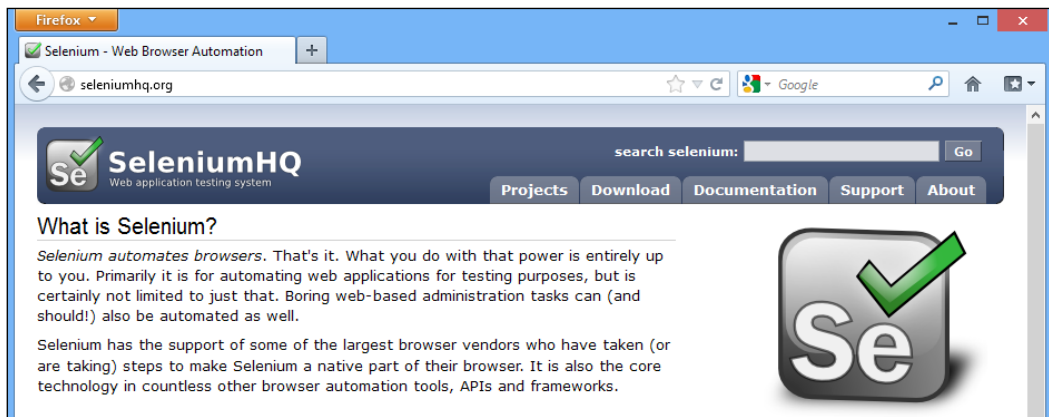
Before you install Selenium IDE, you will need to check that you have all of the required elements, as listed:

- ◆ Disk space: 500 MB free (minimum).
- ◆ Memory: 1 GB (minimum).
- ◆ Selenium IDE requires Mozilla Firefox web browser to be installed on the system.
- ◆ For developing tests with Selenium WebDriver, you will need Eclipse IDE installed on the system. You can download and install Eclipse from <http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/junosr2>.

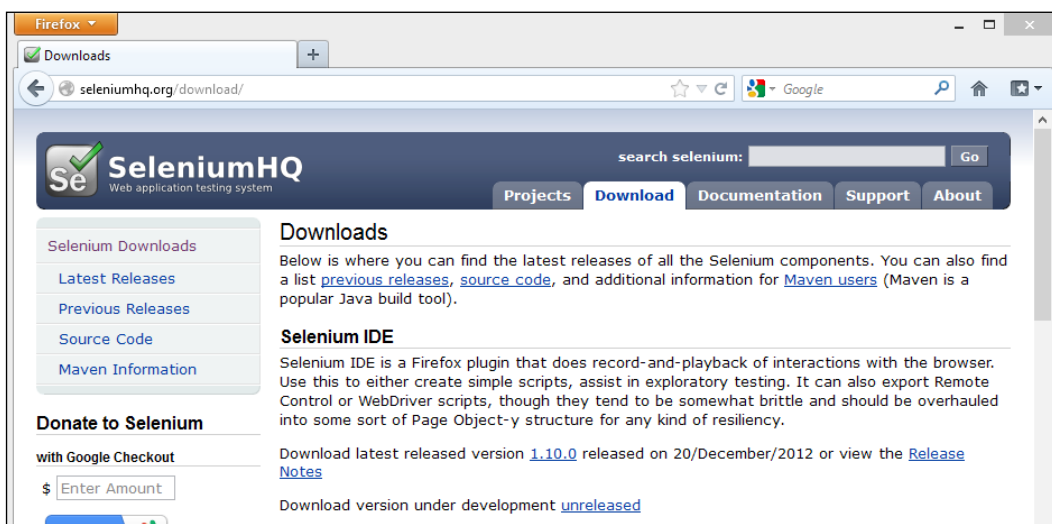
Step 2 – Downloading Selenium IDE

The easiest way to download Selenium IDE as a Firefox add-on is from <http://seleniumhq.org>.

1. Start the Firefox browser and navigate to the Selenium home page at <http://seleniumhq.org> and click on the **Download** tab as shown in the following screenshot:



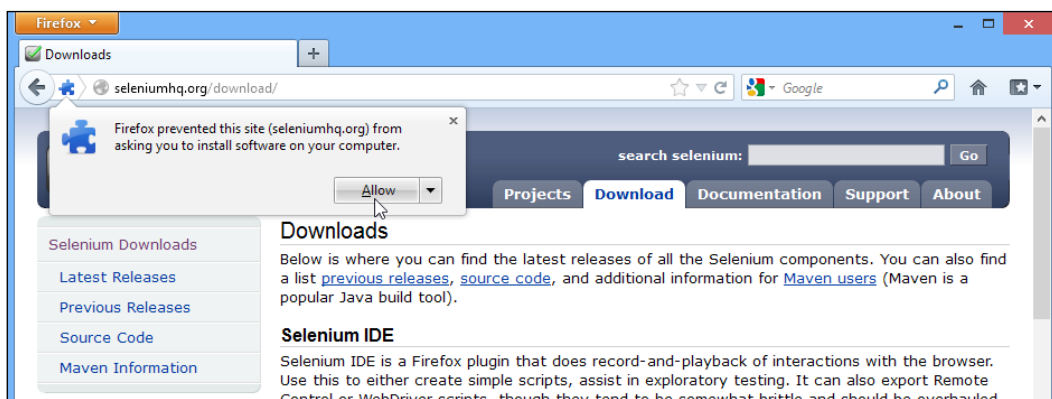
2. On the **Download** page, go to the **Selenium IDE** section and click on the version link as shown in the following screenshot. We suggest that you download the most current stable build and as of writing this book the most stable build is 1.10.0:



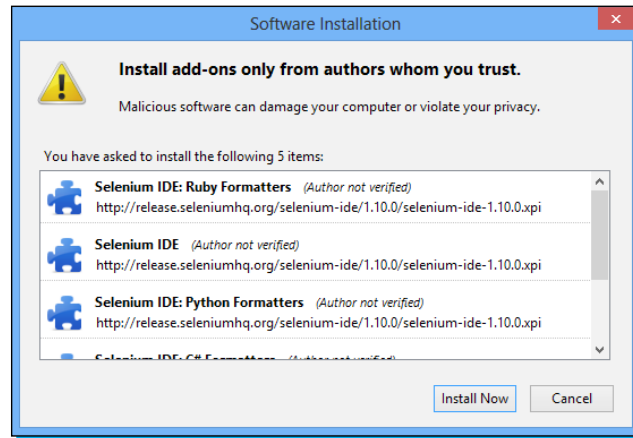
Step 3 – Installing Selenium IDE

Installing Selenium IDE is quick and easy, similar to installing any other add-on in Firefox. Selenium IDE Version 1.10.1 is supported on Firefox Version 19:

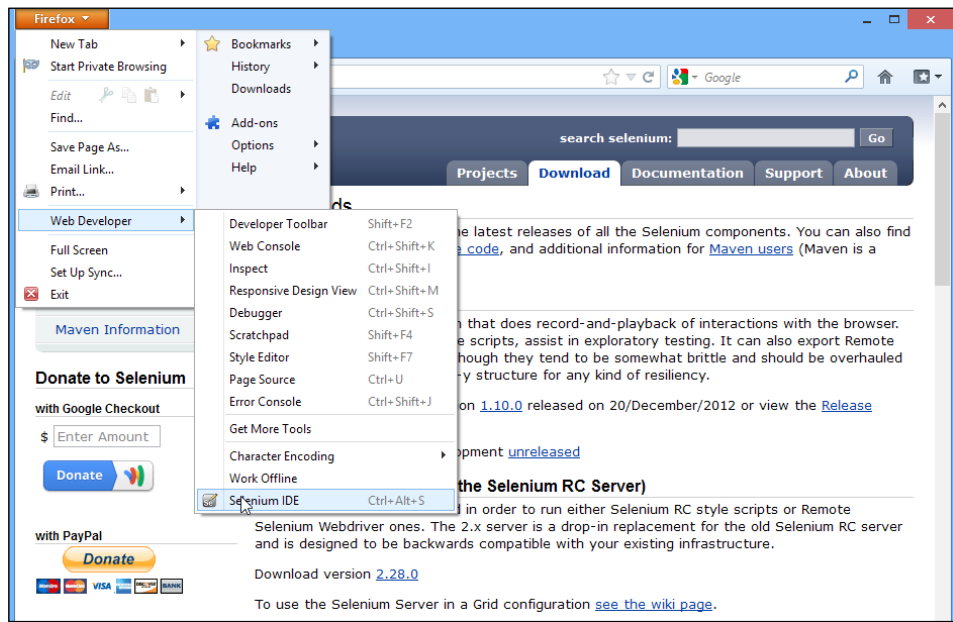
1. After downloading, Firefox will request your permission to install the add-on. Click on the **Allow** button as shown in the following screenshot:



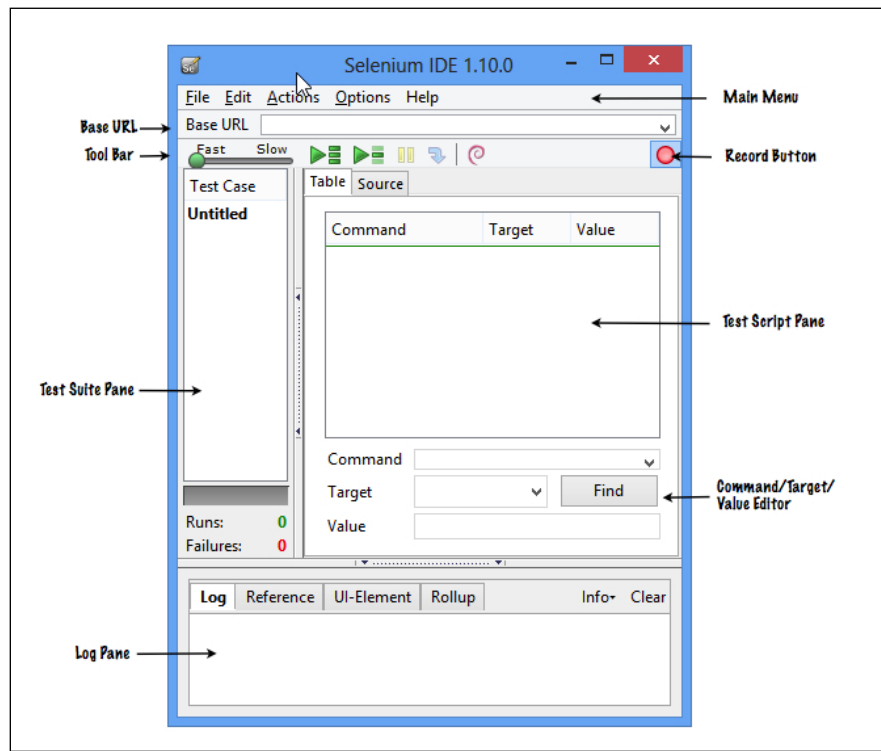
- Firefox will download the add-on and display the **Software Installation** dialog box as shown in the following screenshot. Click on the **Install Now** button:



- Firefox will install the Selenium IDE. Firefox will restart after the add-on is installed.
- To launch the Selenium IDE, click on **Firefox | Web Developer | Selenium IDE** as shown in the following screenshot. You might see a Firefox menu in some cases (on operating systems such as Linux, Ubuntu, and Mac OS X), where you can open the Selenium IDE by clicking on **Tools | Selenium IDE** from the Firefox main menu:



5. Selenium IDE will be launched on top of the Firefox window as shown in the following screenshot:



And that's it!

By this point, you should have a working installation of Selenium IDE, and you should be free to play around and discover more about it. Installing Selenium IDE is as simple as installing any other add-on or extension for Firefox. In the next section, you will see the power Selenium IDE has and how useful it is for automating and testing web applications.


Quick start

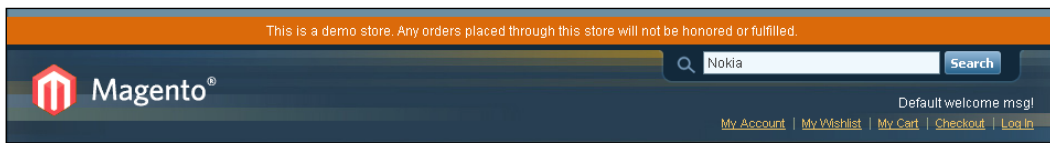
In this section we will show you how to record a test using Selenium IDE. During the recording, we will add some additional commands to the test and run the recorded test. At the end of this section, we will show you how to export the recorded test to Selenium WebDriver.

A test is a basic building block in Selenium IDE. It contains commands for navigation, test steps, and checks for expected versus the actual state of the application. In this section we will show you how to create your first test and execute this test with Selenium IDE.

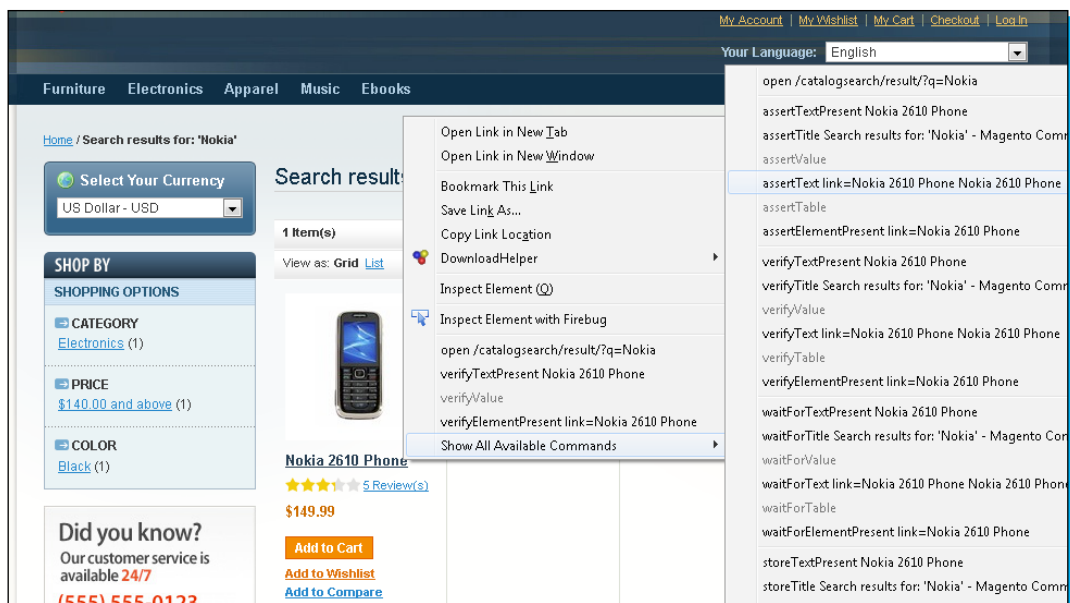
Step 1 – Recording and adding commands in a test

In this section we will show you how to record a test on a demo e-commerce application. We will test the product search feature of the application using the following steps:

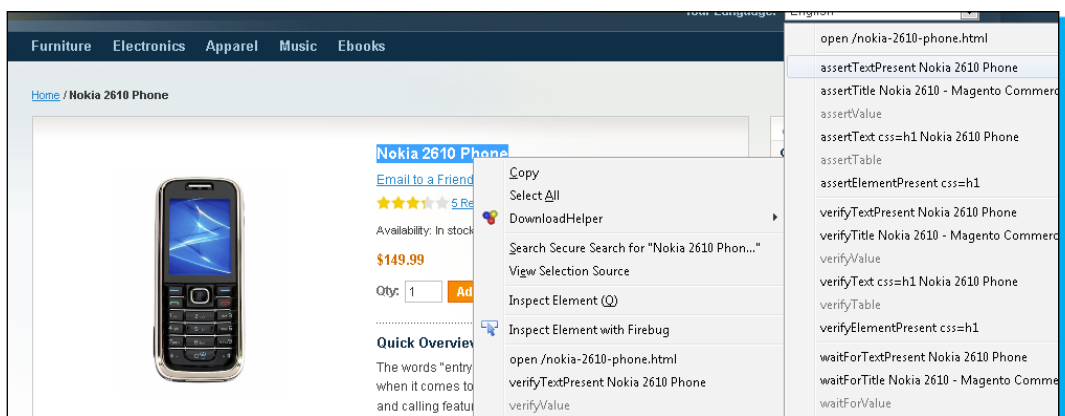
1. Launch the Firefox browser.
2. Open the website for testing in the Firefox browser. For this example we will use `http://demo.magentocommerce.com/`.
3. Open Selenium IDE from the **Tools** menu.
4. Selenium IDE by default sets the recording mode on. If it's not pressed, you can start recording by pressing the  (record) button in the top-right corner.
5. Now switch back to the Firefox browser window and type `Nokia` in the search textbox and click on the **Search** button as shown:



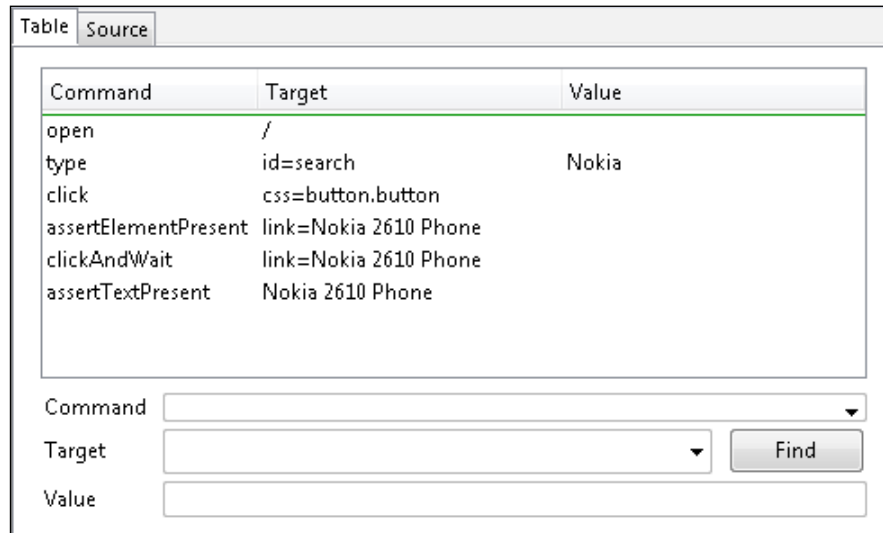
6. Check if the link **Nokia 2610 Phone** is present in the search results. We can do that by selecting the link and opening the context menu (right-click) and selecting **Show All Available Commands** | `assertElementPresent link=Nokia 2610 Phone`.



- Next, we will click on the **Nokia 2610 Phone** link to open the product page and check if the **Nokia 2610 Phone** text is displayed on the product page. To do this, select the **Nokia 2610 Phone** text and open the context menu (right-click) and select **Show All Available Commands | assertTextPresent link=Nokia 2610 Phone**:



- Go back to Selenium IDE. All the previous steps are recorded by Selenium IDE in the **Command-Target-Value** format as shown in the following screenshot. Stop the recording session by clicking on the **Recording** button:

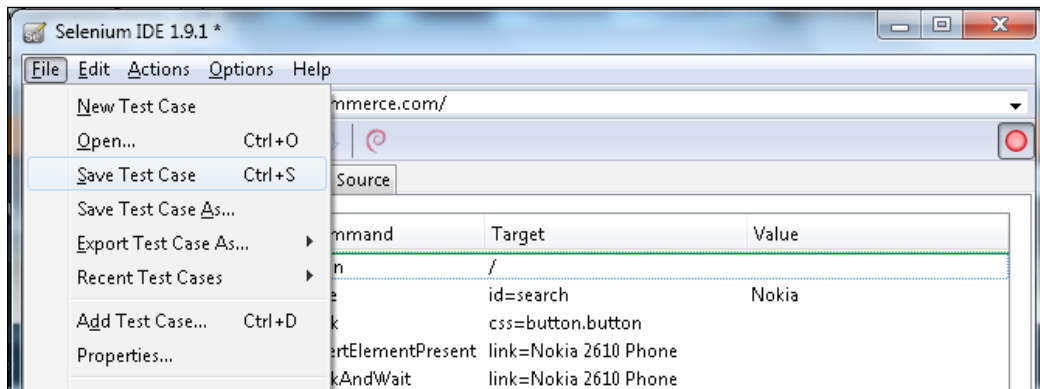


Command	Target	Value
open	/	
type	id=search	Nokia
click	css=button.button	
assertElementPresent	link=Nokia 2610 Phone	
clickAndWait	link=Nokia 2610 Phone	
assertTextPresent	Nokia 2610 Phone	

Step 2 – Saving the recorded test

Before we play back the recorded test, let's save it in Selenium IDE:

- Select **File | Save Test Case** from the Selenium IDE main menu:



- In the **Save As** dialog box, enter the test case name as `SearchTest.html` and click on the **Save** button. The test will be saved with the name **SearchTest**.

Step 3 – Saving the test suite


In Selenium IDE, we can group multiple tests in a test suite. Let's create a test suite and Selenium IDE will automatically add `SearchTest` to this suite:

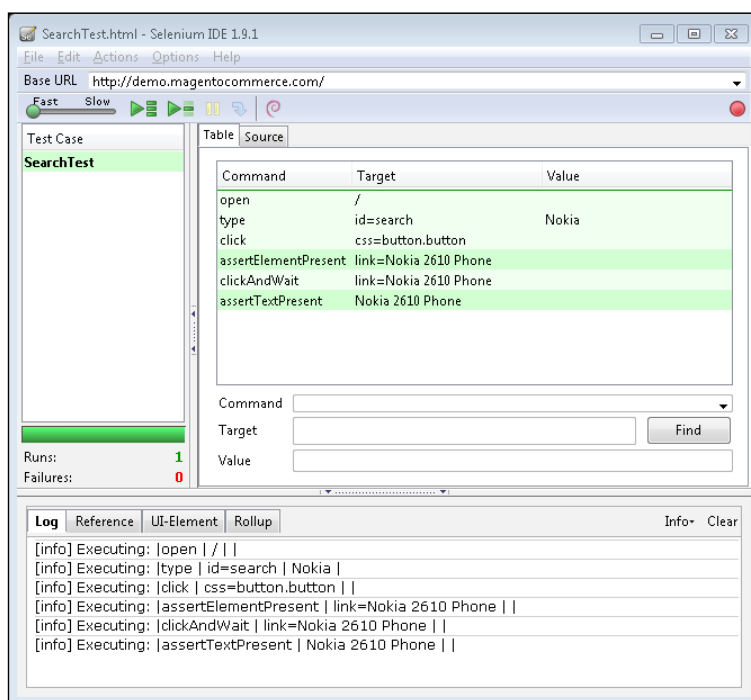
1. Select **File | Save Test Suite** from the Selenium IDE main menu.
2. In the **Save As** dialog box, enter the test case name as `SearchFeatureTests.html` and click on the **Save** button.
3. You can create and record more than one test case in a test suite.

Step 4 – Running the recorded test


Selenium IDE provides multiple ways to execute the tests:

◆ Option 1 – running a single test case

1. Select the test which you want to execute from the test suite pane and click on the  (play current test case) button.
2. Selenium IDE will start the playback of the test and you can see the steps that we recorded earlier are being played automatically in the browser window. At end of execution, Selenium IDE will display results as per the following screenshot:



◆ **Option 2 – running all tests from a test suite**

If you have multiple tests in a test suite, you can use the  (play the entire test suite) button to play all the test cases.

After the test is executed in Selenium IDE, you can see the results in Log tab. All the steps which are successfully completed will be highlighted in green and checks in dark green. If there are any failures in the test, those will be highlighted in red. This is how Selenium IDE helps you testing your web application.

Step 5 – Exporting a recorded test to Selenium WebDriver

Selenium IDE is great to start, however, you will need more to build a test automation framework. Complex tests, with conditional statements and logic, parameterization cannot be automated very well using Selenium IDE. In addition to this you might need features such as detailed logging and reporting, error handling particularly unexpected errors, database testing, and integration with other testing tools and frameworks that are not supported by Selenium IDE.

For building a flexible, maintainable, and robust framework, you will need to use Selenium WebDriver.

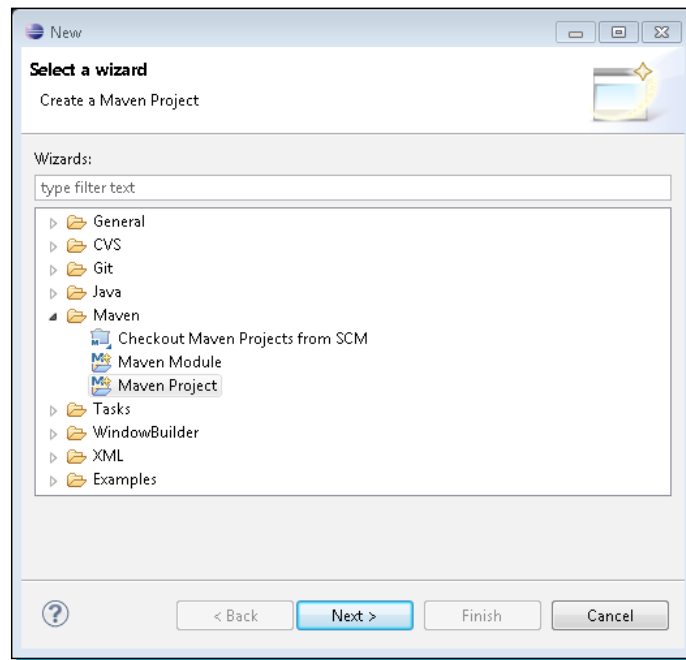
Selenium WebDriver is a simple API that helps you with browser automation. However, when using it for testing and building a test framework, there is much more needed. You will need to integrate Selenium WebDriver API with different libraries, tools, and so on, for test development. You will need an **integrated development environment (IDE)** to build your test project and inject other dependencies in to the framework.

We will show how to use Eclipse for developing tests with Selenium WebDriver. Eclipse is a very popular IDE in the Java world. Eclipse provides a feature-rich environment for Selenium WebDriver test development in Java. Along with Eclipse, Apache Maven also provides support for managing the entire lifecycle of a test project. Maven is used to define project structure, dependencies, build, and test management.

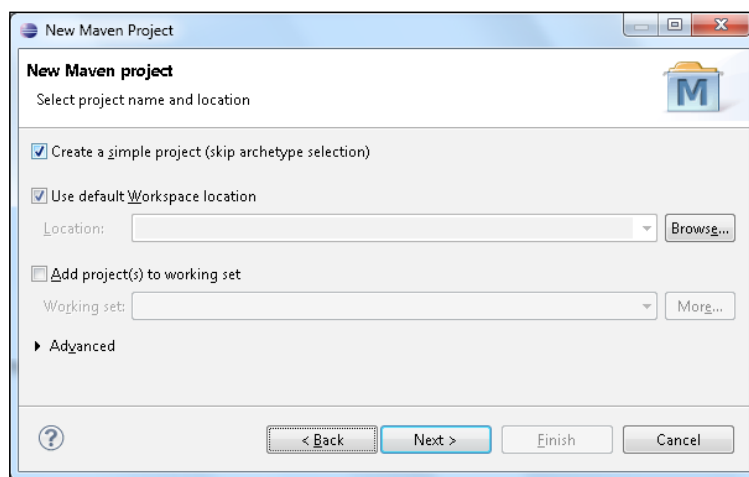
You can use Eclipse and Maven for building your Selenium WebDriver test framework from a single window. Another important benefit of using Maven is that you can get all the Selenium library files and their dependencies by configuring the `pom.xml` file. Maven automatically finds and downloads the dependencies to the project from the Maven central repository while building the project. In this section we will show you how to configure a new Maven project in Eclipse with the following steps:

1. Launch the Eclipse IDE.
2. Create a new project by selecting **File | New | Other** from the **Eclipse** main menu.

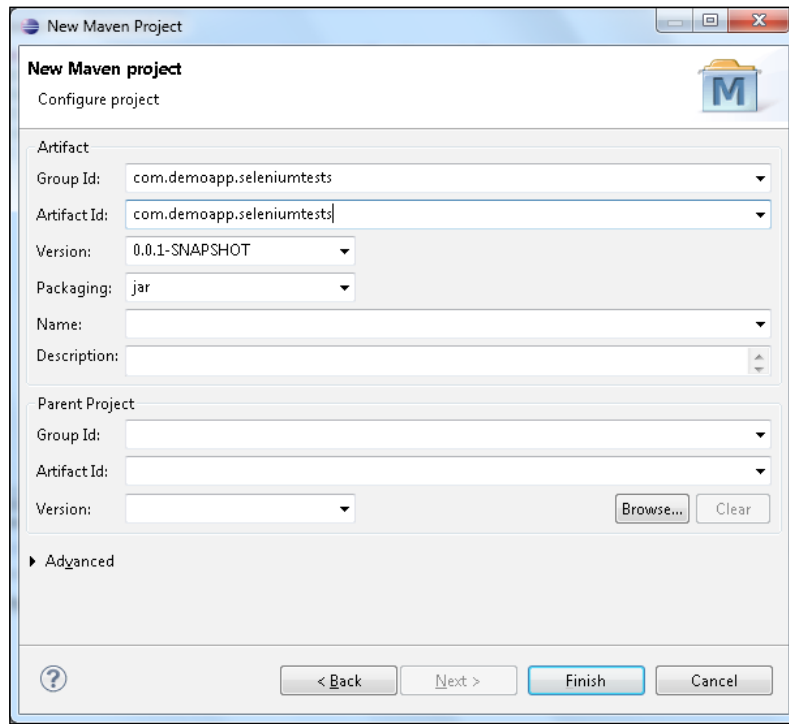
3. On the **New** dialog, select **Maven | Maven Project** as shown in the following screenshot:



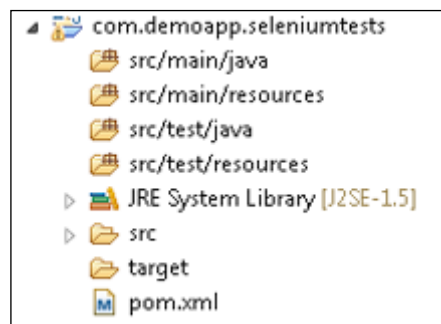
4. Next, the **New Maven Project** dialog box will be displayed; select the **Create a simple project (skip archetype selection)** checkbox and keep everything as default and click on the **Next** button:



5. On the **New Maven Project** dialog box, enter `com.demoapp.seleniumtests` in the **Group Id** and **Artifact Id** textboxes. You can also add a name and description. Keep everything as default and click on the **Finish** button, as shown in the following screenshot:



6. Eclipse will create the `com.demoapp.seleniumtests` project with a folder structure (in **Package Explorer**) similar to the one shown in the following screenshot:



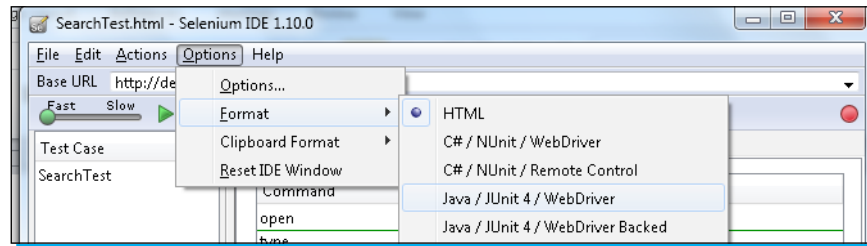
7. Select **pom.xml** from **Package Explorer**. This will open the `pom.xml` file in the editor area with the **Overview** tab open. Select the **pom.xml** tab instead.

8. Add the WebDriver and JUnit dependencies highlighted in the following code to **pom.xml** in the `<project>` node:

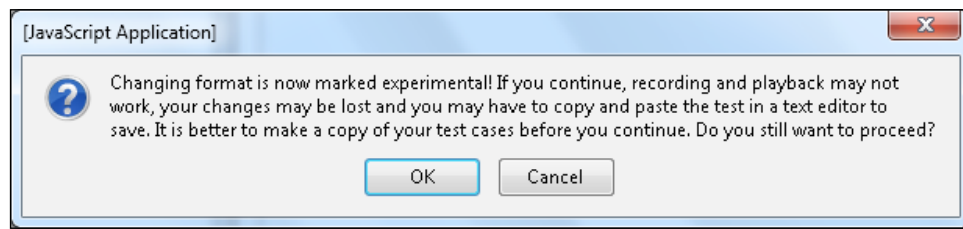
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.demoapp.seleniumtests</groupId>
  <artifactId>com.demoapp.seleniumtests</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>LATEST</version>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

9. You can get the latest dependency information for Selenium WebDriver and JUnit from <http://seleniumhq.org/download/maven.html> and <http://maven.apache.org/plugins/maven-surefire-plugin/examples/junit.html> respectively.
10. Add the **com.example.tests** package in the `src/test/java` folder.
11. Create a new class `SearchTest.java` in the **com.example.tests** package.
12. Now let's go back to Selenium IDE and click on **Options | Options** from the main menu.
13. In the **Selenium IDE Options** dialog box, check the **Enable experimental features** checkbox and click on the **OK** button.

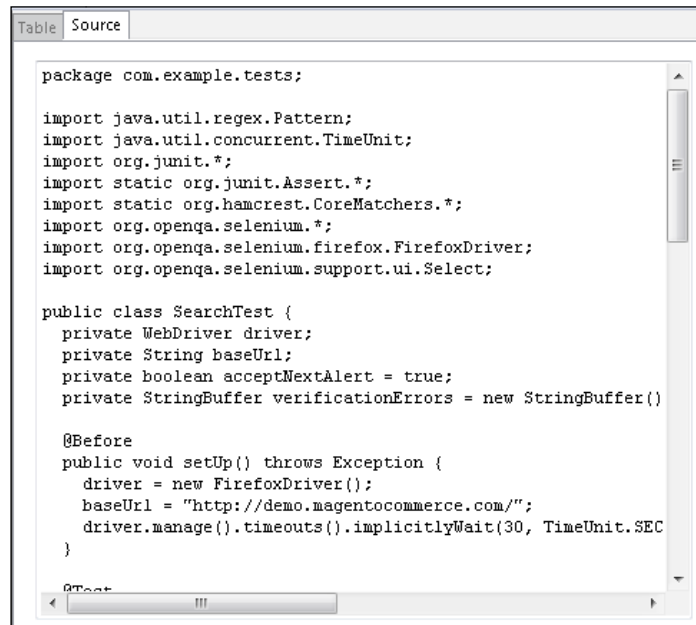
- To view the test in Java, select **Options | Format | Java / JUnit 4 / WebDriver** from the main menu as shown in the following screenshot:



- Click on the **OK** button in the **[JavaScript Application]** dialog box as shown:



- The **Source** tab on the Selenium IDE window will now display the test in Java:



17. Copy the entire contents from the **Source** tab and add it to the SearchTest class:

```
package com.example.tests;

import java.util.regex.Pattern;
import java.util.concurrent.TimeUnit;
import org.junit.*;
import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;

public class SearchTest {
    private WebDriver driver;
    private String baseUrl;
    private boolean acceptNextAlert = true;
    private StringBuffer verificationErrors = new StringBuffer();

    @Before
    public void setUp() throws Exception {
        driver = new FirefoxDriver();
        baseUrl = "http://demo.magentocommerce.com/";
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.
SECONDS);
    }

    @Test
    public void testSearch() throws Exception {
        driver.get(baseUrl + "/");
        driver.findElement(By.id("search")).clear();
        driver.findElement(By.id("search")).sendKeys("Nokia");
        driver.findElement(By.cssSelector("button.button")).click();
        assertTrue(isElementPresent(By.linkText("Nokia 2610 Phone")));
        driver.findElement(By.linkText("Nokia 2610 Phone")).click();
        // Warning: assertTextPresent may require manual changes
        assertTrue(driver.findElement(By.cssSelector("BODY")).
getText().matches("^\\s\\s\\s*Nokia 2610 Phone\\s\\s\\s*$"));
    }

    @After
    public void tearDown() throws Exception {
        driver.quit();
        String verificationErrorString = verificationErrors.

```

```
toString();
    if (!"".equals(verificationErrorString)) {
        fail(verificationErrorString);
    }
}

private boolean isElementPresent(By by) {
    try {
        driver.findElement(by);
        return true;
    } catch (NoSuchElementException e) {
        return false;
    }
}

private String closeAlertAndGetItsText() {
    try {
        Alert alert = driver.switchTo().alert();
        if (acceptNextAlert) {
            alert.accept();
        } else {
            alert.dismiss();
        }
        return alert.getText();
    } finally {
        acceptNextAlert = true;
    }
}
```

18. To run the tests in the Maven lifecycle, select the **com.demoapp.seleniumtests** project in **Package Explorer**. Right-click on the project name and select **Run As | Maven test**. Maven will execute all the tests from the project using Selenium WebDriver with other language bindings.

Apart from Java, Selenium WebDriver bindings are available for a number of other programming languages:

◆ Ruby language bindings

You can use Selenium with Ruby by installing the `selenium-webdriver` gem with the following command:

```
gem install selenium-webdriver
```

For more information on Ruby bindings, see <http://code.google.com/p/selenium/wiki/RubyBindings>.

◆ Python language bindings

For using Selenium with Python, use the following command:

```
pip install selenium
```

For more information on Python bindings, see <http://code.google.com/p/selenium/wiki/PythonBindings>.

◆ .NET bindings

You can also use Selenium along with .NET by adding Selenium's NuGet package to the project in Visual Studio. For more information, refer to the bonus chapter *Integration with other tools* from Packt Publishing's *Selenium Testing Tools Cookbook* at http://www.packtpub.com/sites/default/files/downloads/Integration_with_Other_Tools.pdf.

Top 5 features you'll want to know about

As you start using Selenium, you will realize that there are a wide variety of things that you can do with it. In this section, you will learn about the 5 most important Selenium IDE and WebDriver programming features that you will certainly want to know about. Here, you will learn how to work with running tests on different types of web browsers, locating elements, working with WebElements, synchronization, and capturing screenshots during test runs.

1 – Running tests on various browsers

One of the key features of Selenium is support for running tests on a variety of web browsers available. You can develop a test on one browser and run it on all the supported browsers as needed. This feature is essential for cross-browser testing of your web application. You can run tests on combinations of operating systems and web browsers.

◆ Option 1 – running Selenium IDE tests on various browsers

In the *Quick start* section, we created a test in Selenium IDE; however, as we saw, it only allows us to record and replay tests in Firefox. We can use the Selenium standalone server to run tests that are created with Selenium IDE on different browsers other than Firefox using the following steps:

1. Download the latest release of Selenium standalone server from <http://code.google.com/p/selenium/downloads/list>.
2. Open a command prompt/console window and type the following command:

```
java -jar selenium-server-standalone-2.29.0.jar -htmlSuite
*googlechrome http://demo.magentocommerce.com/ C:\SearchTests.
html C:\Result.html
```



We have used `*googlechrome` in the preceding command to run the tests in Google Chrome. You can also use `*iexplore` (Internet Explorer), `*safari` (Safari), and `*firefox` (Firefox) to run tests on these browsers.

This command will launch the Selenium standalone server. We used the `-htmlSuite` option through which we can tell the server to execute the tests created using Selenium IDE. We also specified the browser on which we want to run the tests and the name of the test suite along with the path and name for the output result file. The server will launch Google Chrome and execute all the tests from the specified test suite.

◆ Option 2 – running Selenium WebDriver tests on various browsers

Selenium WebDriver implements support for various web browsers through driver classes, which provide driver specific functionality to the tests. The following list is of the major drivers supported by WebDriver:

◦ Firefox

`FirefoxDriver` is widely used and is a mature driver supporting most of the WebDriver core APIs including HTML5. Firefox driver is supported on major OS platforms including Windows, Linux, and Mac OS X.

Language	Syntax
Java	<code>WebDriver driver = new FirefoxDriver();</code>
C#	<code>IWebDriver driver = new FirefoxDriver();</code>
Ruby	<code>driver = Selenium::WebDriver.for :firefox</code>
Python	<code>driver = webdriver.Firefox()</code>

For more information on `FirefoxDriver`, see <http://code.google.com/p/selenium/wiki/FirefoxDriver>.

◦ Google Chrome

`ChromeDriver` is supported by installing a standalone `ChromeDriver` server, which uses a JSON wire protocol to communicate between the Chrome browser and your test. `chromeDriver` is supported on major OS platforms including Windows, Linux, and Mac OS X. Chrome has certain minor support limitations and does not support the HTML5 API.

Language	Syntax
Java	<code>WebDriver driver = new ChromeDriver();</code>
C#	<code>IWebDriver driver = new ChromeDriver();</code>
Ruby	<code>driver = Selenium::WebDriver.for :chrome</code>
Python	<code>driver = webdriver.Chrome()</code>

For more information on `ChromeDriver`, see <http://code.google.com/p/selenium/wiki/ChromeDriver>.

- Internet Explorer

InternetExplorerDriver is now a standalone server, which implements WebDriver's wire protocol. The driver supports running 32-bit and 64-bit versions of the browser on Windows.

Language	Syntax
Java	<code>WebDriver driver = new InternetExplorerDriver();</code>
C#	<code>IWebDriver driver = new InternetExplorerDriver();</code>
Ruby	<code>driver = Selenium::WebDriver.for :ie</code>
Python	<code>driver = webdriver.Ie()</code>

For more information on InternetExplorerDriver, see <http://code.google.com/p/selenium/wiki/InternetExplorerDriver>.

- Safari

SafariDriver is added to the list of supported browsers recently. It is implemented as a Safari browser extension instead of the client/server model used in ChromeDriver and InternetExplorerDriver. It communicates with the WebDriver client using WebSockets supported on the Mac OS X and Windows platforms.

Language	Syntax
Java	<code>driver = new SafariDriver();</code>
C#	<code>IWebDriver driver = new SafariDriver();</code>
Ruby	<code>driver = Selenium::WebDriver.for :safari</code>
Python	<code>driver = webdriver.Remote("http://localhost:4444/wd/hub ", webdriver.DesiredCapabilities.SAFARI)</code>

For information on SafariDriver, see <http://code.google.com/p/selenium/wiki/SafariDriver>.

- Opera

OperaDriver is developed by Opera Software and its community. It is supported on all major OS platforms and is available as a core API and standalone server.

Language	Syntax
Java	<code>WebDriver driver = new OperaDriver();</code>
C#	<code>IWebDriver driver = new RemoteWebDriver(new Uri("http://localhost:4444/wd/hub"), DesiredCapabilities.Opera());</code>
Ruby	<code>driver = Selenium::WebDriver.for :opera</code>
Python	<code>driver = webdriver.Opera()</code>

For more information on OperaDriver, see <http://code.google.com/p/selenium/wiki/OperaDriver>.

- iPhone/iPad

iPhoneDriver allows testing web applications on iOS using a special application that uses UIWebView (a WebKit browser accessible for third-party applications) on iOS devices. This is done by installing and running a iWebDriver app on the iOS device or simulator.

Language	Syntax
Java	<code>WebDriver driver = new iPhoneDriver();</code>
C#	<code>IWebDriver driver = new RemoteWebDriver(new Uri("http://localhost:3001/wd/hub"), DesiredCapabilities.IPhone());</code>
Ruby	<code>driver = Selenium::WebDriver.for :iphone</code>
Python	<code>driver = webdriver.Remote("http://localhost:3001/wd/hub", webdriver.DesiredCapabilities.IPHONE)</code>

For more information on iPhoneDriver, see <http://code.google.com/p/selenium/wiki/IDriver>.

- Android

AndroidDriver allows testing of web applications on the Android browser. Similar to iPhoneDriver, an Android application running on the device or emulator is used to run the tests.

Language	Syntax
Java	<code>WebDriver driver = new AndroidDriver();</code>
C#	<code>IWebDriver driver = AndroidDriver();</code>
Ruby	<code>driver = Selenium::WebDriver.for :android</code>
Python	<code>driver = webdriver.Remote("http://localhost:3001/wd/hub", webdriver.DesiredCapabilities.ANDROID)</code>

For more information on `AndroidDriver`, see <http://code.google.com/p/selenium/wiki/AndroidDriver>.

- `RemoteWebDriver`

`RemoteWebDriver` is a critical component of Selenium that allows testing on browsers located on remote machines. You can use `RemoteWebDriver` for running your tests in a distributed architecture. The `RemoteWebDriver` consists of a client and server. The server is simply a Java Servlet running within the Jetty Servlet container. This servlet interacts with the various browsers. In the beginning of this section we saw `selenium-server-standalone`, which is the `RemoteWebDriver` server. The client is an instance of `RemoteWebDriver`, which communicates with the server via a JSON wire protocol similar to other drivers. We can specify what configuration is needed for testing by using `DesiredCapabilities`. For example we need to test an application on an iPad, and we can configure the `RemoteWebDriver` in the following way to run the test:

```
WebDriver driver = new RemoteWebDriver(new URL("http://localhost:3001/wd/hub"), DesiredCapabilities.ipad());
```

For more information on `RemoteWebDriver`, see <http://code.google.com/p/selenium/wiki/RemoteWebDriver>.

2 – Locating elements

One of the key features of Selenium is its ability to locate different types of elements used on a page and the ability to interact with them. Selenium provides various ways or strategies to locate elements and perform actions such as clicking, typing text, selecting an option from the element, or performing verification on the state or property of the element, such as a text value. We can also check if the element is enabled or disabled, is checked or unchecked.

- ◆ **Selenium IDE:** For locating elements using Selenium IDE, you need to specify locator details in the **Target** field.
- ◆ **Selenium WebDriver:** For locating elements using WebDriver, the `driver` interface provides the `findElement()` and `findElements()` methods, which take locator expressions and search for the matching element(s).

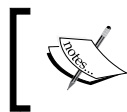
The following table describes some of the important locator strategies that you can use with IDE or WebDriver API:

Strategy	Description	Selenium IDE	Selenium WebDriver (Java)
identifier=<id>	Locates the first element that has the specified value for the @id attribute. IDs are uniquely assigned by developers for key elements on a page. This strategy is considered the most reliable and faster.	id=search	<code>driver.findElement(By.id("search"))</code>
name=<name>	Locates the first element with the specified value for the @name attribute.	name=search	<code>driver.findElement(By.name("search"))</code>
link=<text Pattern>	Locates the link (anchor) element that matches the specified pattern in link text.	link=Nokia 2610 Phone	<code>driver.findElement(By.linkText("Checkout"))</code> or <code>driver.findElement(By.partialLinkText("Inbox"))</code> (You can specify a partial link text instead of complete link text. This might be useful when links partially contain dynamic values.)
css=<cssSelector Syntax>	Locates elements using CSS selectors (refer to http://goo.gl/a9v7T and http://goo.gl/c1IRf for more details). CSS selectors are useful when the above selectors do not work to identify elements uniquely. You can also use XPath as an alternative, but CSS is considered faster with respect to strategy as compared to XPath.	<code>css=input.search</code> (this will locate input element whose @class attribute has value 'search')	<code>driver.findElement(By.cssSelector("input.search"))</code>

Strategy	Description	Selenium IDE	Selenium WebDriver (Java)
xpath=<xpath Expression>	Locates elements using an XPath query. XPath is used to query XML documents and while HTML is subset of XML and browsers represent HTML documents as XHTML. We can use this strategy to locate elements.	xpath=//input [@class='search']	driver.findElement(By.xpath("//input [@class='search']"))
dom=<javascript Expression>	This strategy uses JavaScript expressions to find elements from the document object model (DOM) .	dom=document.getElementById("search")	NA

In addition to the aforementioned strategies, Selenium WebDriver supports the following strategies:

Strategy	Description	WebDriver API (Java) example
By class name	Locates the first element that has the specified value for the @class attribute	driver.findElements(By.className("search"))
By tag name	Locates elements using their HTML tag	For example, we can locate all the link (anchor) elements from a page with driver.findElements(By.tagName("a"))



Refer to Packt Publishing's *Selenium Testing Tools Cookbook* (<http://www.packtpub.com/recipes-to-master-selenium-2-testing-tools-cookbook/book>) for more information on locators.

3 – Working with HTML elements

Selenium provides an extensive support for the standard HTML elements used on a web page. You can interact with the HTML elements by using built-in commands and APIs for building simple to complex tests. The following table shows some key commands and API methods for interacting with a page and its elements:

Purpose	Selenium IDE	Selenium WebDriver (Java)
Click on an element	click	<code>element.click()</code>
Type a text	type	<code>element.sendKeys()</code>
Check a checkbox/ radio button	check	<code>If(!element.isSelected() { element.click() })</code>
Uncheck Checkbox/ Radio Button	uncheck	<code>If(element.isSelected() { element.click() })</code>
Select item(s) in a list or drop-down list	select addSelection (for multi select)	<code>element. selectByVisibleText("Option"); or element. selectByVisibleValue("Option"); or element.selectByVisibleIndex(1);</code>
Remove selection from a list or a drop- down list	removeSelection removeAllSelection	<code>element.deselectByVisibleText("O ption"); or element.deselectByVisibleValue("O ption"); or element. deselectByVisibleIndex(1);</code>
Get the inner text from the element	storeText	<code>element.getText()</code>
Get attribute value	-	<code>element.getAttribute("attribute")</code>

Selenium WebDriver provides the `WebElement` class for interacting with HTML elements.

4 – Synchronizing steps

When Selenium scripts are played back, the application may not always respond with the same speed, especially for applications using AJAX. For example, it might take a few seconds for the following:

- ◆ To load page contents
- ◆ For a window or pop up message to open
- ◆ For a progress bar to reach 100 percent
- ◆ For a status message to appear
- ◆ For a button to become enabled

You can handle these anticipated timing problems by synchronizing your script to ensure that Selenium waits until your application is ready before performing a certain step. There are several options that you can use to synchronize your script using Selenium IDE and Selenium WebDriver.

Selenium IDE

Selenium IDE provides various built-in `waitFor` commands for handling synchronization problems in tests. The following is a list of some `waitFor` commands:

Command	Condition
<code>waitForPageToLoad</code>	Delays execution until the page is fully loaded in the browser
<code>waitForElementPresent</code>	Delays execution until the specified element is present on the page
<code>waitForElementNotPresent</code>	Delays execution until the specified element is removed from the page
<code>waitForTextPresent</code>	Delays execution until the specified text is present on the page
<code>waitForFrameToLoad</code>	Delays execution until the contents of the frame are fully loaded in the browser
<code>waitForAlertPresent</code>	Delays execution until an alert window is displayed on the page

A number of these commands are run implicitly when other commands are being executed. For example, with respect to the `clickAndWait` command, when you click on an element, the `waitForPageToLoad` command is also executed.

Selenium WebDriver

Selenium WebDriver provides implicit and explicit wait conditions to handle synchronization problems. You can use these conditions in the following ways:

◆ Option 1 – the implicit wait condition

When an implicit wait condition is implemented, if Selenium WebDriver cannot find an element in the DOM, it will wait for a defined amount of time for the element to appear in the DOM. In other terms, an implicit wait condition polls the DOM for a certain amount of time when trying to find an element(s) if it is not immediately available. The default setting is 0. Once set, the implicit wait condition is set for the life of the WebDriver object's instance. Here is a sample test using the implicit wait condition:

```
@Test
public void testWithImplicitWait()
{
    //Go to the Demo AJAX Application
    WebDriver driver = new FirefoxDriver();
    driver.get("http://dl.dropbox.com/u/55228056/AjaxDemo.html");

    //Set the Implicit Wait time Out to 10 Seconds
    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

    try {
        //Get link for Page 4 and click on it
        WebElement page4button = driver.findElement(By.
linkText("Page4"));
        page4button.click();

        //Get an element with id page4 and verify it's text
        WebElement message = driver.findElement(By.id("page4"));
        assertTrue(message.getText().contains("Nunc nibh tortor"));
    } catch (NoSuchElementException e) {
        fail("Element not found!!");
        e.printStackTrace();
    } finally {
        driver.close();
    }
}
```

However, an implicit wait condition may slow down your tests when an application responds normally, as it will wait for each element appearing in the DOM and increase the overall execution time. It is recommended to avoid or minimize the use of an implicit wait condition.



Minimize or avoid using an implicit wait condition in your scripts and try to handle synchronization issues with an explicit wait condition, which provides more control as compared to an implicit wait condition.

◆ Option 2 – the explicit wait condition

The explicit wait condition provides a better control compared with an implicit wait condition. Unlike an implicit wait condition, you can write custom code or conditions for a wait before proceeding further in the code. An explicit wait condition can only be implemented in cases where synchronization is needed and the rest of the script is working fine.

The Selenium WebDriver provides the `WebDriverWait` and `ExpectedCondition` classes for implementing an explicit wait condition. You can use these classes in the following way:

```
@Test
public void testExplicitWaitTitleContains()
{
    //Go to the Google Home Page
    WebDriver driver = new FirefoxDriver();
    driver.get("http://www.google.com");

    //Enter a term to search and submit
    WebElement query = driver.findElement(By.name("q"));
    query.sendKeys("selenium");
    query.click();

    //Create Wait using WebDriverWait.
    //This will wait for 10 seconds for timeout before title is
    updated with search term
    //If title is updated in specified time limit test will move to
    the text step
    //instead of waiting for 10 seconds
    WebDriverWait wait = new WebDriverWait(driver, 10);
    wait.until(ExpectedConditions.titleContains("selenium"));

    //Verify Title
    assertTrue(driver.getTitle().toLowerCase().
    startsWith("selenium"));
    driver.quit();
}
```

The `ExpectedCondition` class provides a set of predefined conditions to wait before proceeding further in the code. The following table shows some common conditions that we frequently come across when automating web browsers supported by the `ExpectedCondition` class:

Predefined condition	Selenium WebDriver (Java)
An element is visible and enabled	<code>elementToBeClickable(By locator)</code>
An element is selected	<code>elementToBeSelected(WebElement element)</code>
Presence of an element	<code>presenceOfElementLocated(By locator)</code>
Specific text present in an element	<code>textToBePresentInElement(By locator, java.lang.String text)</code>
Element value	<code>textToBePresentInElementValue(By locator, java.lang.String text)</code>
Title	<code>titleContains(java.lang.String title)</code>



For more conditions, visit <http://selenium.googlecode.com/svn/trunk/docs/api/java/org/openqa/selenium/support/ui/ExpectedConditions.html>.

5 – The Page Object pattern

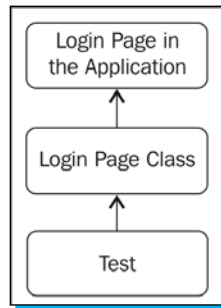
The Page Object pattern provides tests for an interface, where a test can operate on the logical functionality offered by the page in a manner similar to the user accessing the page, but by hiding its internals. For example, if we build a Page Object for a login page that will provide a method to log in by accepting the username and password, and will take the user to the home page of the application. The test need not worry about what type of input controls are used for the login page, their locator details, navigation, and so on.

Tests should use objects of a page at a high level, where any change in layout or attributes used for the fields in the underlying page should not break the test.

Selenium WebDriver provides outstanding support for implementing the Page Object pattern via its `PageFactory` class. The Page Object pattern brings the following advantages for your tests:

- ◆ It helps in building a layer of abstraction separating automation code, which knows about locating application elements and the one which interacts with these elements for actual testing
- ◆ It provides a central repository of pages from the application for tests
- ◆ It provides high maintainability and reduction in code duplication

The login page class provides an interface to the login page of the application to the tests as shown in the following diagram:



Here is Page Object for the login page of the test application.

The login page class contains locator details for the key elements needed for user login functionality. Elements from the login page are defined as private members of the LoginPage class. The test code will not have access to these elements. The LoginPage class provides a public login() method to the tests. The test needs to pass the e-mail address and password for a registered user to this method. The constructor of the LoginPage class uses the PageFactory.initElements() method to initialize WebElements defined in the class in the following way:

```
package demo.magentocommerce.pages;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

public class Login {

    @FindBy(id="email")
    private WebElement emailField;

    @FindBy(id="pass")
    private WebElement passwordField;

    @FindBy(id="send2")
    private WebElement loginButton;

    public Login(WebDriver driver) {
        PageFactory.initElements(driver, this);
    }
}
```

```
public void login(String email, String password) {  
    emailField.sendKeys(email);  
    passwordField.sendKeys(email);  
    loginButton.click();  
}  
}
```

Here is a sample test using the LoginPage class. The instance of the LoginPage class is created by passing the current driver instance. The login() method is called in the following way:

```
...  
//Navigate to the Login Page  
WebDriver driver = new FirefoxDriver();  
driver.get("http://demo.magentocommerce.com/customer/account/login/");  
  
//Create an instance of Login page and call the Login function  
LoginPage loginPage = new LoginPage(driver);  
loginPage.login("xxx@xx.com", "xxxx");  
  
//Verify the user is logged in  
...
```



Refer to Packt Publishing's *Selenium Testing Tools Cookbook* for more information on Page Object pattern's advanced techniques.

People and places you should get to know

If you need help with Selenium, here are a few people and places which will prove to be invaluable:

Official sites

- ◆ Home page: <http://www.seleniumhq.org> and <http://code.google.com/p/selenium>
- ◆ Manual and documentation: <http://seleniumhq.org/docs/>
- ◆ Wiki: <http://code.google.com/p/selenium/wiki/>
- ◆ Blog: <http://seleniumhq.wordpress.com/>
- ◆ Source code: <https://code.google.com/p/selenium/source/checkout>

Articles and tutorials

- ◆ *An Introduction To Selenium IDE*, available at <http://blog.softwaretestingclub.com/2011/03/an-introduction-to-selenium-ide/>
- ◆ *Data Driven Testing with Selenium IDE*, available at <http://unmesh.me/2012/12/04/data-driven-testing-with-selenium-ide/>
- ◆ *Using WebDriver*, available at <http://code.google.com/p/selenium/wiki/UsingWebDriver>
- ◆ *How to use WebDriverWait*, available at <https://blog.mozilla.org/webqa/2012/07/12/how-to-webdriverwait/>
- ◆ *XPath, CSS, DOM and Selenium: The Rosetta Stone* available at <http://www.simple-talk.com/dotnet/.net-framework/xpath,-css,-dom-and-selenium-the-rosetta-stone/>

Community

- ◆ Official mailing list: selenium-users@googlegroups.com
- ◆ Official forums: <https://groups.google.com/forum/?fromgroups#!forum/selenium-users>
- ◆ Unofficial forums: <http://sqa.stackexchange.com/> and <http://www.sqaforums.com/forums/forumdisplay.php?f=72&s=b671fbd058460956e6f50e44989e24d0>
- ◆ Official IRC channel: #selenium at Freenode (irc.freenode.net)
- ◆ User FAQ: <http://code.google.com/p/selenium/wiki/FrequentlyAskedQuestions>

Blogs

- ◆ The blog of Adam Goucher at <http://element34.ca/?cat=blog> & <http://adam.goucher.ca/?p=1331>
- ◆ The blog of David Burns at <http://www.theautomatedtester.co.uk/>
- ◆ Alister Scott at <http://watirmelon.com/>
- ◆ The official Sauce Labs blog at <http://saucelabs.com/>
- ◆ You can also get a list of all Selenium bloggers at http://it-kosmopolit.de/Selenium/blog/selenium-blogs/selenium_blogs.php

Twitter

- ◆ Follow Selenium HQ on Twitter at <http://twitter.com/seleniumhq> for latest updates and announcements
- ◆ Simon Stewart: <http://twitter.com/shs96c>
- ◆ David Burns: <http://twitter.com/automatedtester>
- ◆ Adam Goucher: <http://twitter.com/adamgoucher>
- ◆ Sauce Labs: <https://twitter.com/saucelabs>
- ◆ For more open source information, follow Packt at <http://twitter.com/#!/packtopensource>



Thank you for buying **Instant Selenium Testing Tools Starter**

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

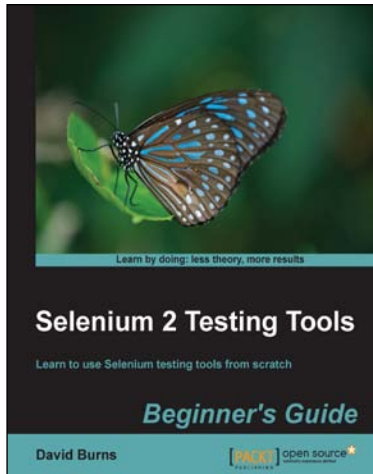
Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

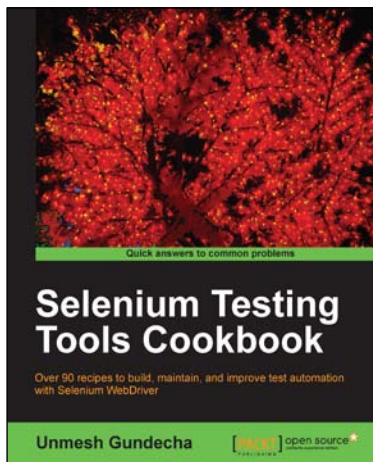


Selenium 2 Testing Tools: Beginner's Guide

ISBN: 978-1-84951-830-7 Paperback: 232 pages

Learn to use Selenium Testing tools from scratch

1. Automate web browsers with Selenium WebDriver to test web applications.
2. Set up Java Environment for using Selenium WebDriver.
3. Learn good design patterns for testing web applications.



Selenium Testing Tools Cookbook

ISBN: 978-1-84951-574-0 Paperback: 326 pages

Over 90 recipes to build, maintain and improve test automation with Selenium WebDriver

1. Learn to leverage the power of Selenium WebDriver with simple examples that illustrate real world problems and their workarounds.
2. Each sample demonstrates key concepts allowing you to advance your knowledge of Selenium WebDriver in a practical and incremental way.
3. Explains testing of mobile web applications with Selenium Drivers for platforms such as iOS and Android.

Please check www.PacktPub.com for information on our titles

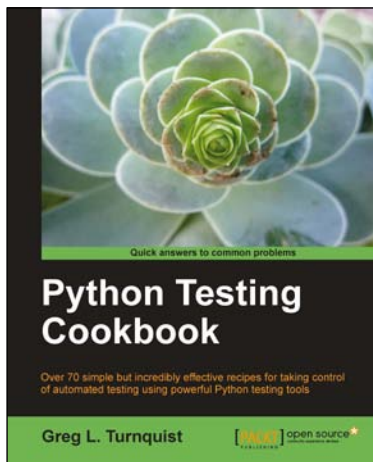


Instant Pygame for Python Game Development How-to

ISBN: 978-1-78216-286-5 Paperback: 76 pages

Create engaging and fun games with Pygame, Python's Game development library

1. Learn something new in an Instant! A short, fast, focused guide delivering immediate results.
2. Quickly develop interactive games by utilizing features that give you a great user experience.
3. Create your own games with realistic examples and easy to follow instructions .



Python Testing Cookbook

ISBN: 978-1-84951-466-8 Paperback: 364 pages

Over 70 simple but incredibly effective recipes for taking control of automated testing using powerful Python testing tools

1. Learn to write tests at every level using a variety of Python testing tools.
2. The first book to include detailed screenshots and recipes for using Jenkins continuous integration server (formerly known as Hudson).
3. Explore innovative ways to introduce automated testing to legacy systems.
4. Written by Greg L. Turnquist – senior software engineer and author of Spring Python 1.1.

Please check www.PacktPub.com for information on our titles

