Test Automation Using Selenium WebDriver with Java

Navneesh Garg

- Selenium WebDriver 2.0
- Learn Automation on a Web Based Application
- Real Life Experiences
- Step By Step Instructions
- Interview Questions Based on Selenium

Selenium WebDriver Step By Step Guide

Test Automation Using Selenium WebDriver with Java

By Navneesh Garg

ISBN - 978-0-9922935-1-2

Publisher: AdactIn Group Pty Ltd.

Contents

About the Author

Navneesh Garg

Navneesh Garg is a recognized test automation architect and corporate trainer, specializing in test automation, performance testing, security testing and test management. As a tool specialist, he has worked on a variety of functional automation tools including Selenium, HP QTP/UFT, TestComplete, TestPartner, SilkTest, Watir, RFT, and on varied technologies including Web, Java, Dot-net, SAP, Peoplesoft and Seibel.

His previous book "Test Automation using Unified Functional Testing" is among the bestselling books on HP QTP. This book has consistently ranked among the top 100 testing books on Amazon. It was the first book to be released globally on the latest version of HP QTP.

He is an entrepreneur and founder of several successful IT companies which encompass the AdactIn Group, CresTech Software, and Planios Technologies.

As an experienced corporate trainer, he has trained professionals in Selenium and other test tools across a wide range of global clients such as Macquarie Bank, Corporate Express, Max New York Life, Accenture, NSW Road and Maritime Services, Australian Dept of Education, HCL Technologies, Sapient, Fidelity Group, Adobe Systems, and many more. He has training experience in diverse geographies such as Australia, India, Hong Kong and USA.

As a technical test delivery head for his company, he has led and managed functional automation testing and performance testing teams across a wide range of domains, using commercial tools and open source tools. Certified in HP QTP, HP Quality Center, HP LoadRunner, IBM Rational Functional Tester and as a Certified Ethical Hacker, he has designed several high-end automation frameworks including using Selenium and its integrations with tools like TestNG, JUnit, Selenium Grid, Jenkins and ANT.

Preface

My motivation for writing this book stems from my hands-on experience in the IT and testing domain and the experience I have gained as an automation consultant working in numerous complex automation projects.

Selenium, being an open source tool, is gaining huge popularity but still is not conceived as an easy to use tool especially by testers due to a variety of reasons, including tool setup, programming background and support issues. A key objective of this book is showcase in a simple guided way how to use Selenium WebDriver so that we can attain maximum return on investment from using the tool. Not only will we learn how to use the tool but also how to effectively create maintainable frameworks using Selenium.

In my previous book "Test Automation using HP Unified Functional Testing" we had taken a similar step by step guided approach using commercial tool HP UFT which has been excellently received by the testing fraternity.

Scope of Topics

As part of the scope of this book we will cover **Selenium WebDriver (Selenium 2.0) with Java** as a programming language. We will also cover how to use **Selenium IDE** which is a Firefox based Selenium Plug-in for easy record and replay.

We will be using **Eclipse** as the main IDE for creating Selenium WebDriver tests.

No prior knowledge of Java language is required for this book but having understanding of object oriented programming language concepts will definitely help. As part of this book we will be covering **Basics of Java** which would be required to use Selenium WebDriver for beginner users.

In the later section we also show how to configure and use **Selenium Grid** to run parallel tests on multiple browsers and OS configurations.

As part of reporting frameworks, the book will show how to configure and use both custom **JUnit and TestNG reports**.

We will also see how Selenium WebDriver integrates with **continuous Integration** tools like **Jenkins.**

My intent in this book is to discuss the key features of Selenium WebDriver, WebDriver methods and cover all crucial aspects of the tool which help to **create effective automation frameworks**.

The book **does not** have samples or examples on how to use Selenium WebDriver with Python, C# and Ruby languages. The book **focuses** on using **Selenium WebDriver with Java language**.

Key Audience

The target audience for this book are manual functional testers who want to **learn Selenium WebDriver quickl**y and who want to create effective automation frameworks that generate positive ROIs to stakeholders.

Salient Features of this Book

This book has been designed with the objective of **simplicity and ease of understanding**.

A major fear amongst functional testers who want to learn Selenium is the fear of programming language and coding. As a part of this we will cover just enough **basics on Java programming language** that will give the readers confidence to use Selenium WebDriver.

This book follows a **unique training based approach** instead of a regular text book approach. Using a step by step approach, it guides the students through the exercises using pictorial snapshots.

Selenium being an open source tool needs quite a few independent components to be installed like Eclipse, TestNG, ANT, etc. This would usually scare testers. In this book we will cover step by step installation and configuration of each of these components.

Another major highlight of this book is a **custom developed Web based application used throughout the book** instead of learning automation on custom html pages with few form fields and links.

Another differentiator is that I have tried to include **many practical examples and issues** which most of the automation testers encounter in day-to-day automation. These experiences will give you an insight into what challenges you could face with automation in the real world. Practical examples cover how to use most of the features within Selenium WebDriver.

It also covers aspects of **Continuous Integration tool; Jenkins** so that Selenium WebDriver scripts can be integrated with the development environment and run on nightly builds.

The book also covers the most **common interview questions** on Selenium WebDriver and automation.

Sample Application and Source Used in Book

The sample application used in the book can be accessed at the following URL:

www.adactin.com/HotelApp/

The source code used in the book can be found at the following link

www.adactin.com/store/

Feedback and Queries

For any feedback or queries you can contact the author at www.adactin.com/contact.html or email navneesh.garg@adactin.com

Order this book

For bulk orders, contact us at orders@adactin.com

You can also place your order online at adactin.com/store/

Acknowledgements

I would like to thank my family (my parents, my wife Sapna, my wonderful kids Shaurya and Adaa) for their continued support. Without them this book would not have been possible.

Special thanks to Emily Jones and William B. for their reviews and feedback, which immensely helped as I worked on this book.

I would also like to thank my colleagues and clients for the inspiration, knowledge and learning opportunities provided.

1

Introduction to Automation

Introduction

In this chapter we will talk about automation fundamentals and understand what automation is and the need for automation. An important objective of this chapter is to understand the economics of automation, and determine when we should carry out automation in our projects. We will also discuss some popular commercial and open source automation tools available in the market.

Key objectives:

- What is automation?
- Why automate? What are the benefits of automation?
- Economics of automation
- Commercial and Open Source automation tools

1.1 What is Functional Automation?

Automation testing is to automate the execution of manually designed test cases without any human intervention.

The purpose of automated testing is to execute manual functional tests quickly and in a cost-effective manner. Frequently, we re-run tests that have been previously executed (also called regression testing) to validate functional correctness of the application. Think of a scenario where you need to validate the username and password for an application which has more than 10,000 users. It can be a tedious and monotonous task for a manual tester and this is where the real benefits of automation can be harnessed. We want to free up manual functional tester's time so that they can perform other key tasks while automation provides extensive coverage to the overall test effort.

When we use the term "automation", there is usually confusion about whether automation scope includes functional and performance testing. Automation covers both.

- Functional Automation – Used for automation of functional test cases in the regression test bed.

- Performance Automation – Used for automation of non-functional performance test cases. An example of this is measuring the response time of the application under considerable (for example 100 users) load.

Functional automation and performance automation are two distinct terms and their automation internals work using different driving concepts. Hence, there are separate tools for functional automation and performance automation.

For the scope of this book, we will be only referring to **Functional Automation**.

1.2 Why do we Automate?

Find below key benefits of Functional Automation:

1. **Effective Smoke (or Build Verification) Testing**

Whenever a new software build or release is received, a test (generally referred to as "smoke test" or "shakedown test") is run to verify if the build is testable for a bigger testing effort and major application functionalities are working correctly. Many times we spend hours doing this only to discover that a faulty software build resulted in all the testing efforts going to waste. Testing has to now start all over again after release of a new build.

If the smoke test is automated, the smoke test scripts can be run by developers to verify the build quality before being released to the testing team.

2. **Standalone - Lights Out Testing**

Automated testing tools can be programmed to kick off a script at a specific time.

If needed, automated tests can be automatically kicked off overnight, and the testers can analyse the results of the automated test the next morning. This will save valuable test execution time for the testers.

3. **Increased Repeatability**

At times it becomes impossible to reproduce a defect which was found during manual testing. Key reason for this could be that the tester forgot which combinations of test steps led to the error message; hence, he is unable to reproduce the defect. Automated testing scripts take the guess work out of test repeatability.

4. **Testers can Focus on Advanced Issues**

As tests are automated, automated scripts can be base-lined and re-run for regression testing. Regression tests generally yield fewer new defects as opposed to testing newly developed features. So, functional testers can focus on analysing and testing newer or more complex areas that have the potential for most of the defects while automated test scripts can be used for regression test execution.

5. Higher Functional Test Coverage

With automated testing a large number of data combinations can be tested which might not be practically feasible with manual testing. We use the term 'Data driven testing' which means validating numerous test data combinations using one automated script.

6. Other Benefits

- **Reliable:** Tests perform precisely the same operations each time they are run, thereby eliminating human error.
- **Repeatable:** You can test how the software reacts under repeated execution of the same operations.
  Programmable: You can program sophisticated tests that bring out hidden information from the application.
- **Comprehensive:** You can build a suite of tests that cover every feature in your application.
- **Reusable:** You can re-use tests on different versions of an application, even if the user-interface changes.
- **Better Quality Software:** Because you can run more tests in less time with fewer resources.
- **Fast:** Automated tools run tests significantly faster than human users.

1.3 When should we Automate? Economics of Automation

Let us take a scenario. If your Test Manager comes up to you and asks whether it is advisable for your company to automate an application, how would you respond?

In this scenario, the manager is interested in knowing if functional automation will deliver the organization a better return on investment (ROI) besides improving application quality and test coverage.

We can determine whether we should automate a given test if we can determine that the cost of automation would be less than the total cost of manually executing the test cases.

For example, if a test script is to run every week for the next two years, automate the test if the cost of automation is less than the cost of manually executing the test 104 times (2 years will have 104 weeks).

Calculating the **Cost of Test Automation**

Cost of Automation = Cost of tool + labor cost of script creation +
labor cost of script maintenance

Automate if:

Cost of automation is lower than the manual execution of those scripts.

The key idea here is to plan for the cost of script maintenance. I have seen a lot of automation projects fail because project managers did not plan for the labor costs involved in script maintenance.

Example

Let me give you an example from my personal experience.

I performed some automation work for one of our investment banking clients. We had a five-member team, which automated almost 3000 test cases in about six months time, which included around total 30 man months of effort. At the end of project, we gave the client's testing team a hand-over of the entire automation suite created by our team. Our recommendation to them was that they would need at least a one or two member team to continuously maintain the scripts. This was because there were still functional changes happening to the application and scripts would need maintenance. But since the client project manager had no budget allocated for this activity; they skipped this advice and continued to execute automation scripts. After the first six months of the 3000 test cases, only 2000 test cases were passing, while the rest started failing. These scripts failures were because script fixes were needed due to application changes. The client team was okay with that and continued to execute those 2000 working test cases, and got rid of the remaining 1000 test cases, which were now executed manually. After another six months, only scripts corresponding to 1000 test cases were passing. So they got rid of another 1000 test cases and started executing them manually. After another six months (1.5 years in total), all the scripts were failing, and testing had to move back to manual functional testing.

In the above real-life scenario, the cost of automation and its benefits could have been reaped, if the client had allocated 1-2 automation testers (could have been part-time) to maintain the scripts and had properly planned and budgeted for it.

1.4 Commercial and Open Source Automation Tools

This section lists some of the popular Commercial and Open Source Automation Tools.

Vendor	Tool	Details
OpenSource (free)	Selenium	Open Source tools and market leader in Open Source segment. Primary for WWeb-based automation. Support C#, Java, Python, and Ruby as programming language.

OpenSource (free)	Watir	Watir stands for "Web application testing in Ruby". It is again primarily for WWeb application automation and uses Ruby as the programming language.
HP	Unified Functional Testing	HP UFT (previous version was called QTP) is the market leader in Test Automation in the commercial tools segment. It uses VBScript as the programming language and its ease of use makes it a tool of choice against other competing tools.
IBM	Rational Functional Tester	IBM Rational Functional tester is another popular test Automation Tool. We can program in VB.net or Java using this tool. Is recommended for technical testers.
Microfocus	SilkTest	Microfocus bought SilkTest from Borland. It is still a very popular automation tool which uses 4Test (propriety) language. Good for technical testers.
Microsoft	VSTP – Coded UI tests	Coded UI tests come with Microsoft Visual studio Ultimate or Premium version. You can program using VB.net or C# as languages of choice. Fairly good for technical testers.
SmartBear	TestComplete	Low cost alternative to other commercial tools with good features for automation. You have the option to program using VBScript, JScript, C++Script, C#Script or DelphiScript language.

4

Introduction to Selenium

Introducing Selenium

Selenium is an Open Source tool for automating browser-based applications. Selenium is a set of different software tools, each with a different approach to support test automation. The tests can be written as HTML tables or coded in a number of popular programming languages and can be run directly in most modern Web browsers. Selenium can be deployed on Windows, Linux, and Macintosh and many OS for mobile applications like iOS, Windows Mobile, and Android.

Among all Open Source tools, Selenium functional testing tool is considered to be a highly portable software testing framework and one of the best tools available in the current market for automation of Web applications.

Key objectives:

1. Understand Selenium Tool Suite
2. Choosing right Selenium Tool for use
3. Requirements for Selenium Setup

4.1 Selenium's Tool Suite

Selenium is not just a single tool but a suite of software, each catering to different testing needs of an organization. **It has four components.**

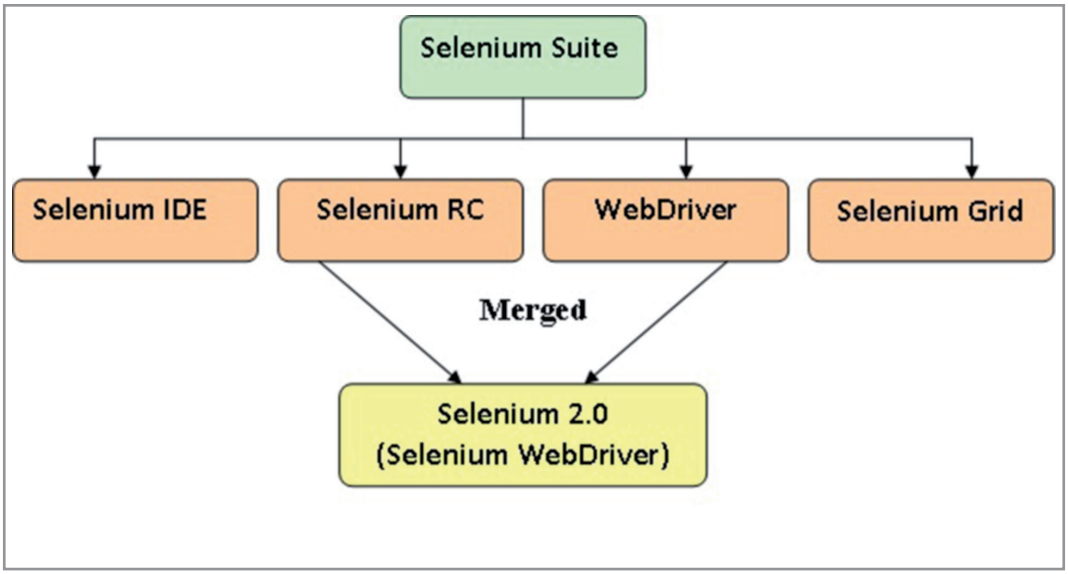


Figure 4.1 - Selenium Suite Structure

In the section below we will understand more about each of these components.

1. **Selenium IDE**

Selenium IDE (Integrated Development Environment) is a prototyping tool for building test scripts. It comes as a Firefox plug-in and provides an easy-to-use interface for developing automated tests. Selenium IDE has a recording feature, which records user actions as they are performed and then exports them as a reusable script in one of many programming languages for execution later.

Selenium IDE is simply intended to be a rapid prototyping tool. Selenium IDE has a "Save" feature that allows users to keep the tests in a table-based format for later import and execution. Selenium IDE doesn't provide iteration or conditional statements for test scripts. Use Selenium IDE for basic automation. Selenium developers usually recommend Selenium 2 or Selenium 1 to be used for serious, robust test automation.

2. **Selenium 1- Selenium RC or Remote Control**

Selenium RC is the main Selenium project allowing user actions to be simulated in a browser like clicking a UI element, input data, etc. It executes the user commands in the browser by injecting Java script functions to the browser when the browser is loaded. As we know, Java Script has its own limitations and so does Selenium RC.

How Selenium RC Works

First, we will describe how the components of Selenium RC operate and the role each plays in running your test scripts.

RC Components: Selenium RC components are:

5

Installing Selenium Components

Introduction

Before we can start using Selenium, there are a few Selenium and non-Selenium components that we need to install. In this chapter we will perform step by step installation and setup of the components which we will need to use over the scope of this book.

> Note: You need access to the internet to download the required setup files.

Key objectives:

1. Setup Instructions for installing Selenium IDE
2. Setup Instructions to install add-on Firebug
3. Setup Instructions to install add-on Firepath
4. Setup Instructions to install Java Development toolkit
5. Setup Instructions to install and setup Eclipse
6. Setup Instructions to install WinANT

5.1 Installing Selenium IDE

Pre-requisite – Firefox browser should be installed locally on the test machine.

1. Launch Firefox browser and open URL **http://seleniumhq.org/download/** to download Selenium IDE from the SeleniumHQ download Page.
2. Click on the latest version of Selenium IDE link within **Selenium IDE** section

> **Note**: The version of the link is constantly being updated. Click on the latest link available at the time you install Selenium IDE.

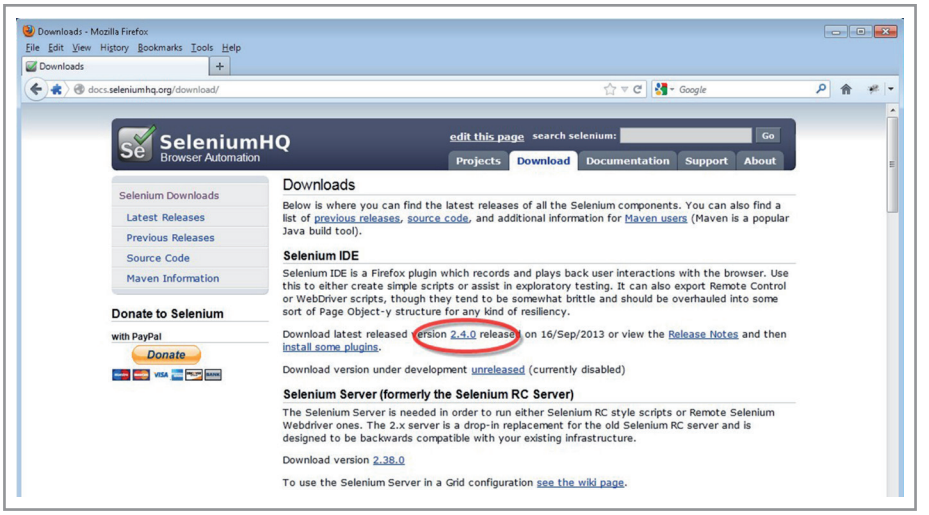


Figure 5.1 – Download Selenium IDE

3. Firefox will protect you from installing add-ons from unfamiliar locations, so you will need to click **'Allow'** to proceed with the installation

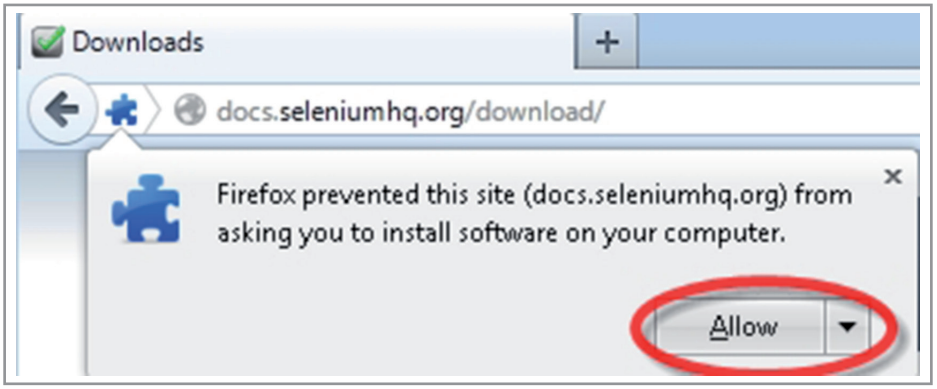


Figure 5.2 – Allow IDE Installation

4. Add-on will get downloaded and you will see Software Installation pop-up. Click on **Install Now**

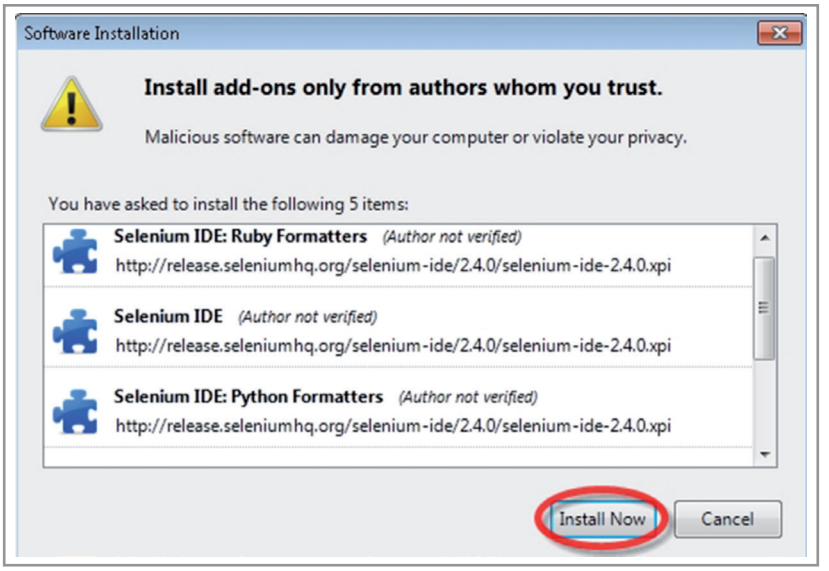


Figure 5.3 – Install Selenium IDE from Firefox Add-on

5. Firefox will show restart dialog to restart Firefox. Click on **Restart Now**

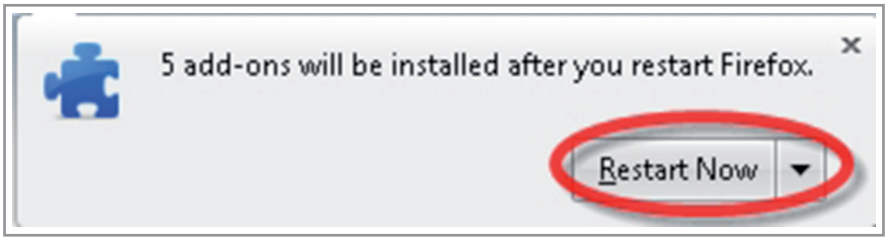


Figure 5.4 - Restart Now Firefox

6. After Firefox reboots you will find the Selenium-IDE listed under the Firefox Tools menu. Go to **Tools → Selenium IDE**

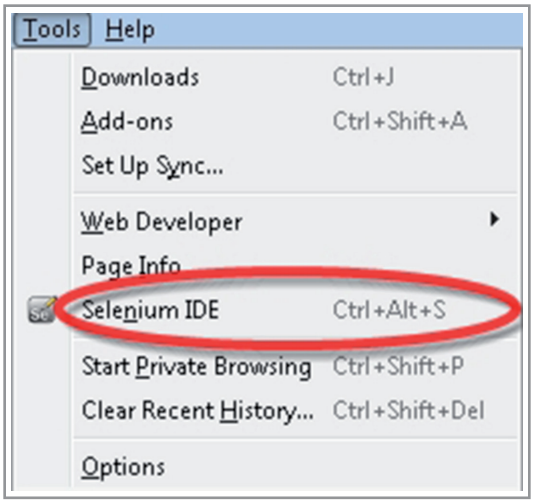


Figure 5.5 – Selenium IDE in Tools Menu

5.2 Installing Firebug plug-in

Now we will install Firefox add-on Firebug (if we haven't done that already).

Firebug integrates with Firefox to give access to Web development tools to edit, debug, and monitor CSS, HTML, and JavaScript live in any Web page.

In Selenium, Firebug helps in inspecting UI elements and finding its associated properties and values.

1. To install Firebug add-ons, Open Firefox browser, launch www.google.com and search for **Firebug.** Click on **Firebug link**.

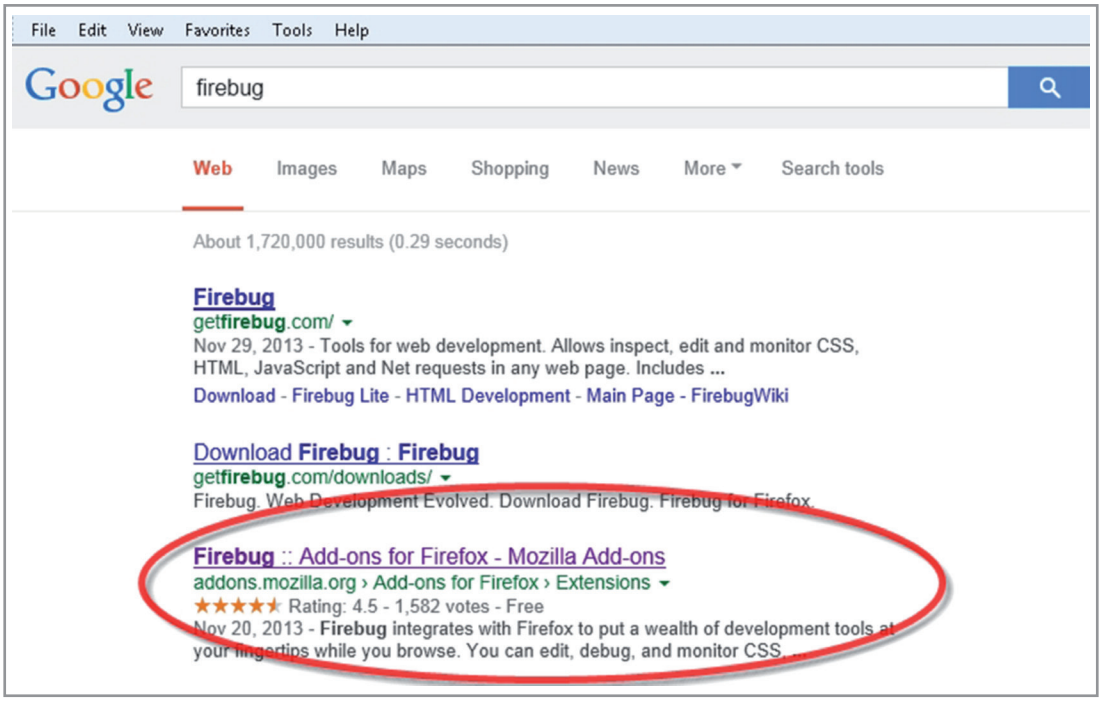


Figure 5.6 Firebug link in Google Search

2. Add-ons page appears. Click on + **Add to Firefox** button

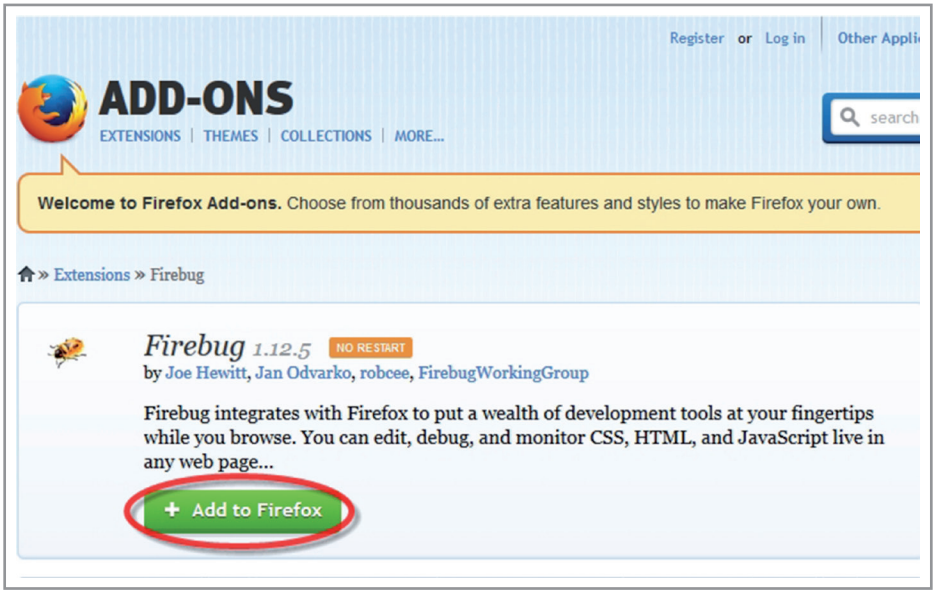


Figure 5.7 – Add Firebug to Firefox

3. Wait for Firebug add-ons to be downloaded. Once downloaded, click on **Install Now** button in Software Installation pop-up.

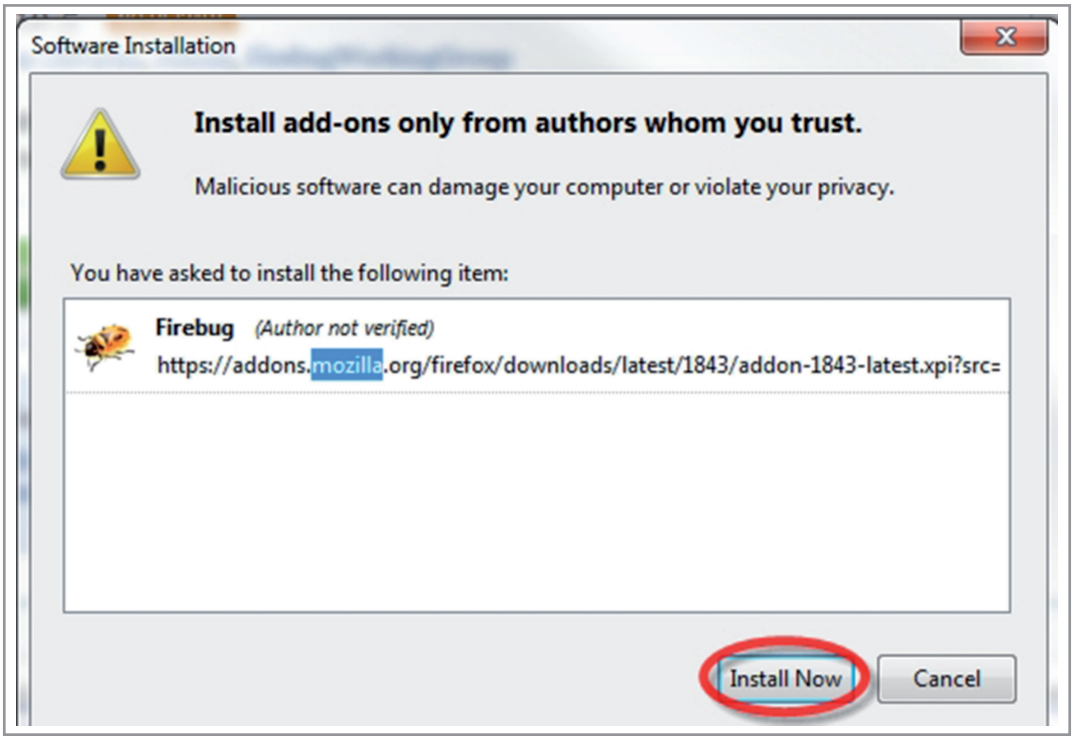


Figure 5.8 – Install Now Firebug

9

Creating First Selenium WebDriver Script

Now that we understand Selenium Basics (Selenium IDE and Locators) and basics of Java we are ready to jump into the real Selenium automation tool – Selenium WebDriver.

In this chapter we will see how to create a WebDriver script. Also, we will configure Eclipse environment.

Key objectives:

- Exporting Selenium IDE script as a Java Selenium WebDriver script
- Configuration of project structure in Eclipse and use Selenium WebDriver script
- Running of Selenium WebDriver script

9.1 Recording and Exporting Script from IDE

In this section we will record the test case using Selenium IDE and then export the test case using ***Java/JUnit 4/WebDriver*** option. Follow the steps given below:

1. Open **Selenium IDE** and verify that recording mode is **ON**
2. Assuming that application login page is already open in **Firefox browser** with login page visible, perform the following steps (in IDE recording mode):
    a. Login (Use the username/password with which you have registered earlier)
    b. Search for a Hotel
        i. Select a location, e.g., Sydney
        ii. Select number of rooms, e.g., 2-Two
        iii. Select adults per rooms, e.g., 2-Two
        iv. Click on Search button
    c. Select a Hotel
        i. Select one of the Hotel Radio buttons, e.g., select radio button next to Hotel Creek
    d. Book a Hotel

i. Enter First Name
ii. Enter Last Name
iii. Enter Address
iv. Enter 16-digit Credit Card No.
v. Enter Credit Card type
vi. Enter Expiry Month
vii. Enter Expiry Year
viii. Enter CVV number
ix. Click on Book Now

e. After you see the Booking confirmation page, click on Logout link in the top right corner
f. Click on "Click here to Login again" link to go back to Home page

3. Stop recording by clicking on **Stop Recording** button in record toolbar
4. Verify the steps below that are recorded Selenium ID

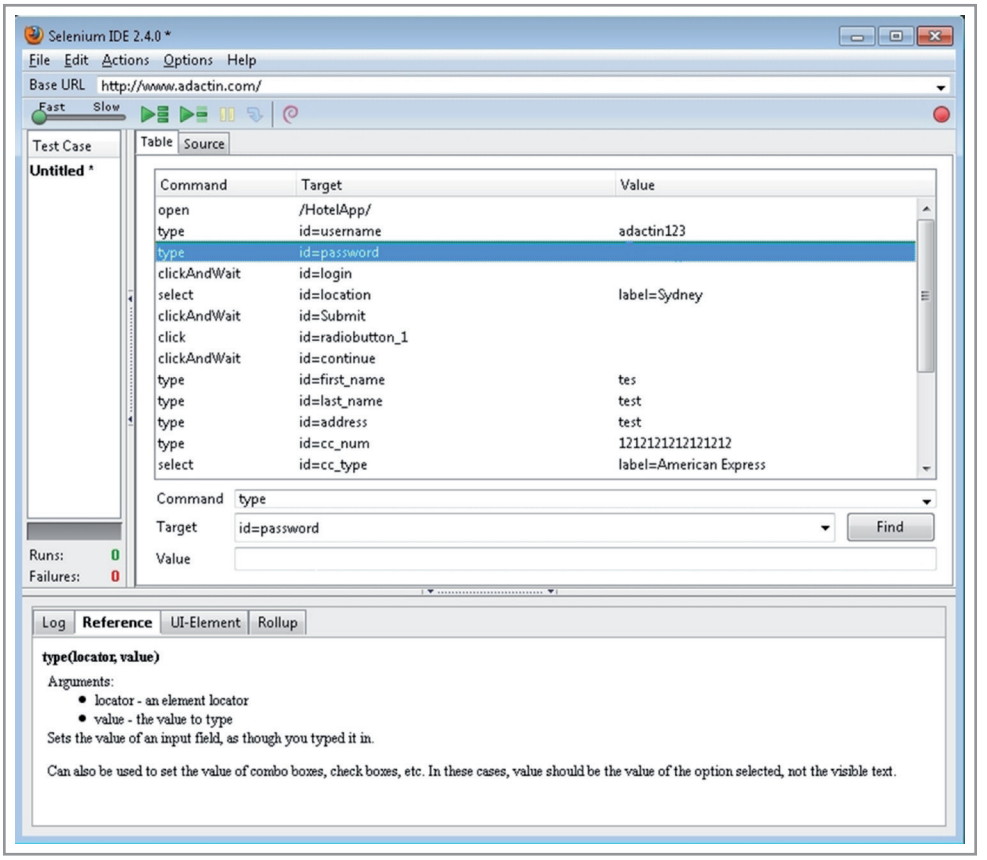


Figure 9.1 IDE Script

1. Select to **File → Export Test Case As → Java/ JUnit 4/WebDriver**

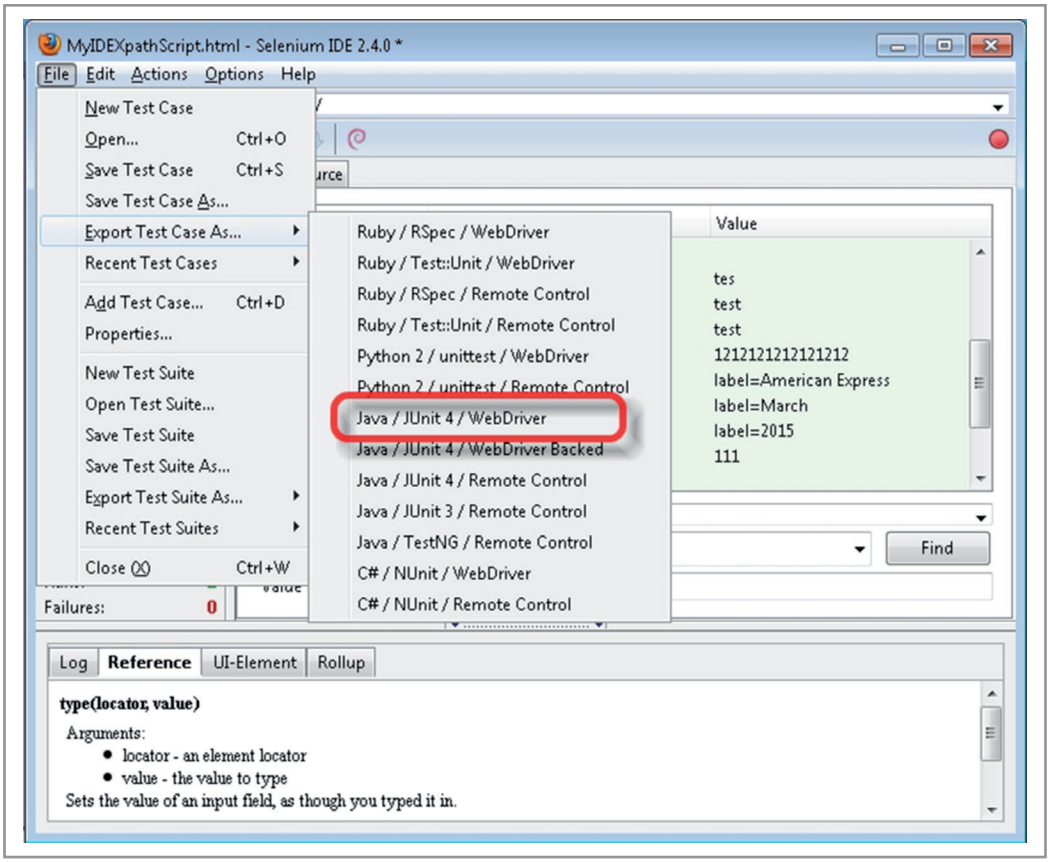


Figure 9.2 Export IDE Test

Note: We will focus on Selenium WebDriver with Java but as you can see Selenium supports export as C#, Python and Ruby languages as well.

2. Save it as MyFirstWebDriverTest in C:\Selenium Folder. You will notice that the script is saved as MyFirstWebDriverTest.java file
3. Try to open the script you have saved in an editor like NotePad++ (you can download this freely from internet)

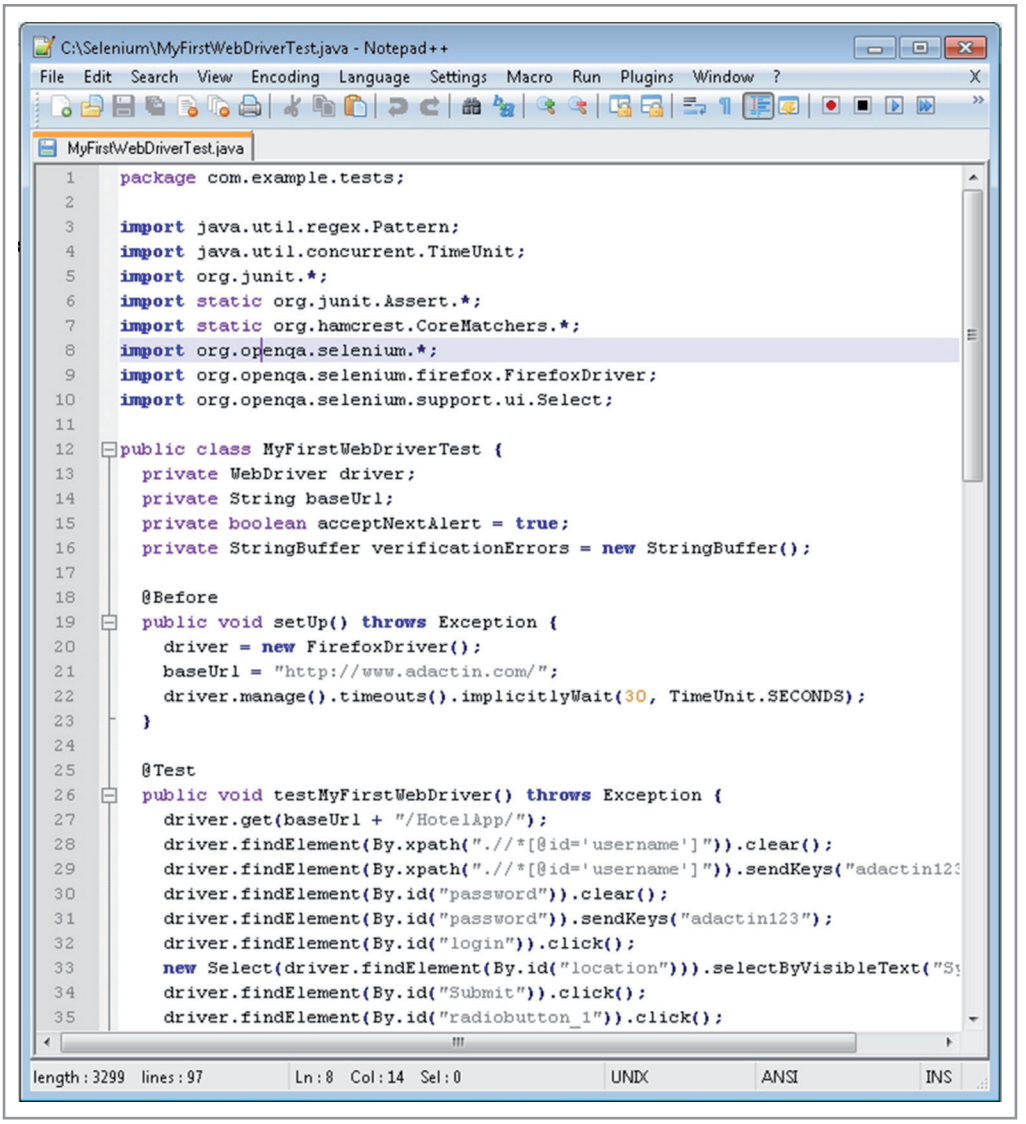


Figure 9.3 - Java code for WebDriver test

Next step will be to review this code and use it in Eclipse.

1. Let us review the exported Java code. The exported test is Junit test. JUnit is a unit testing framework for the Java programming language.
2. We will see a class “MyFirstWebDriverTest” shown in the snapshot below

The highlighted lines in the snapshot below will acquire an instance of a new Firefox browser and assign it to the driver (WebDriver) object which we will use to perform all of our browser actions.

12.4 Assert Statements in Junit

JUnit provides static methods in the Assert class to test for certain conditions. These *assertion methods* typically start with assert and allow you to specify the error message, the expected and the actual result. An *assertion method* compares the actual value returned by a test to the expected value, and throws an AssertionException if the comparison test fails.

When we use the Assert statement we do not have to use an 'If-Else' logical statement as the Assert statement will verify the result for us and return the correct value.

But important to note is that in case of failure, the Assert statement will **Abort** and exit the script. There are ways of implementing the Assert statement to avoid stopping and exiting the script. We will see a sample implementation in the script below.

The following table gives an overview of these methods. Parameters in [] brackets are optional.

Test methods

Statement	Description
fail(String)	Let the method fail. Might be used to check that a certain part of the code is not reached. Or to have a failing test before the test code is implemented. The String parameter is optional.
assertTrue([message], boolean condition)	Checks that the boolean condition is true.
assertFalse([message], boolean condition)	Checks that the boolean condition is false.
assertEquals([String message], expected, actual)	Tests that two values are the same. Note: for arrays the reference is checked, not the content of the arrays.
assertEquals([String message], expected, actual, tolerance)	Test that float or double values match. The tolerance is the number of decimals which must be the same.
assertNull([message], object)	Checks that the object is null.
assertNotNull([message], object)	Checks that the object is not null.
assertSame([String], expected, actual)	Checks that both variables refer to the same object.
assertNotSame([String], expected, actual)	Checks that both variables refer to different objects.

Figure 12.18 – Junit Assert Statements

Let us see an example where we had earlier used the Assert statement.

1. Go to your Selenium IDE and make sure script "IDEVerificationScript" is Open

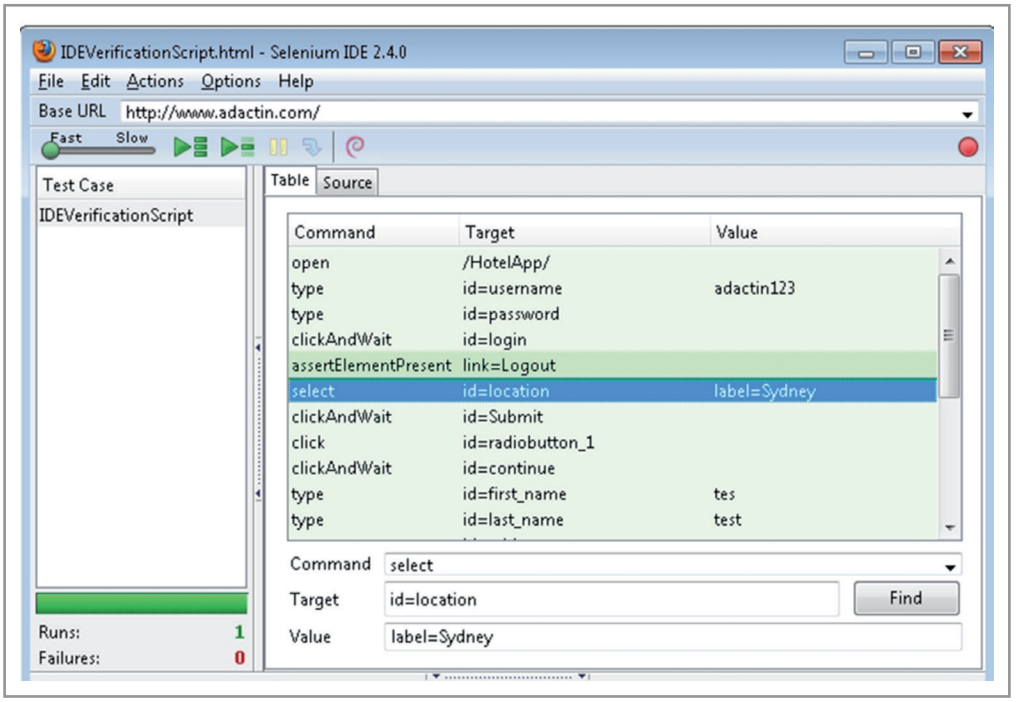


Figure 12.19– Verification Point IDE Script

2. Go to **File → Export Test Case As… → Java/JUnit4/WebDriver** and export the script as a WebDriver Junit test
3. Open the exported WebDriver test in NotePad++

```
@Test
public void testWebDriverVerification() throws Exception {
  driver.get(baseUrl + "/HotelApp/");
  driver.findElement(By.id("username")).clear();
  driver.findElement(By.id("username")).sendKeys("adactin123");
  driver.findElement(By.id("password")).clear();
  driver.findElement(By.id("password")).sendKeys("          ");
  driver.findElement(By.id("login")).click();
  assertTrue(isElementPresent(By.linkText("Logout")));
  new Select(driver.findElement(By.id("location"))).selectByVisibleText("Sydney");
  driver.findElement(By.id("Submit")).click();
  driver.findElement(By.id("radiobutton_1")).click();
  driver.findElement(By.id("continue")).click();
  driver.findElement(By.id("first_name")).clear();
  driver.findElement(By.id("first_name")).sendKeys("tes");
  driver.findElement(By.id("last_name")).clear();
  driver.findElement(By.id("last_name")).sendKeys("test");
```

Figure 12.20 - Exported Verification Point Script

If you notice a new statement, **assertTrue** has been added which validates that the logout link is present.

14

Using Functions

Functions help divide your test into logical units, such as areas of key functionalities of the application. Functions help make our scripts modular and reusable, which will save us maintenance effort and also help us improve productivity. These functions can then be re-used in different scripts.

For example, all of our scripts will have to login to the application. Now, instead of recording login steps repeatedly in every script, we can keep an external login function and re-use that function in all of our scripts.

Example

Let us see another practical example here:

At one of our client engagements, we were automating an investment banking application. As a first step of every test case, we had to create investment instruments after which we had to validate, and add details in later steps (we had more than 100 test cases for each instrument type). Creating an instrument was a tedious step with up to 50 field values to be entered. Based on the test scenario, input data would change. Now recording the steps of investment instrument creation in each and every script would have been a nightmare and time consuming. It would have also been a maintenance issue, if in later development stages the application workflow is changed or new fields were added.

So we created functions to create instruments and for each of the test cases that were automated, we just invoked the same function in every script. This helped us reduce the overall time to automate. This also assisted in maintenance down the line, when the investment instrument creation workflow changed.

Key objectives:

- Create Functions
- Calling Functions in WebDriver script

14.1 Creating Functions in WebDriver

Key steps in creating Functions in WebDriver using Eclipse IDE include:

1. Create a separate Package and Class for Functions

2. Create Function definition and import any required Java libraries in the class
3. Add steps to functions based on function's objective
4. Replace any data within functions with arguments from that function
5. Within the script, import Functions Package and Extend class to use function within your scripts

Pre-conditions

1. Select **HotelApp_TestAutomation/src** folder, right click and select **New → Package**

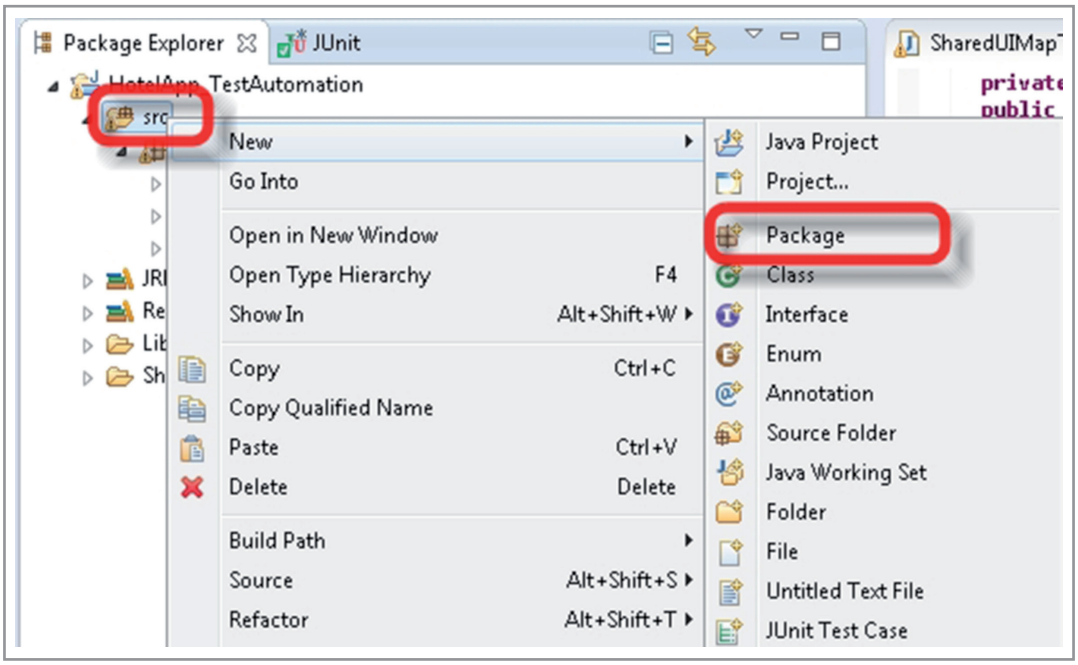


Figure 14.1 – New Package Creation

2. In Java Package dialog enter the Name **functions** and click **Finish**

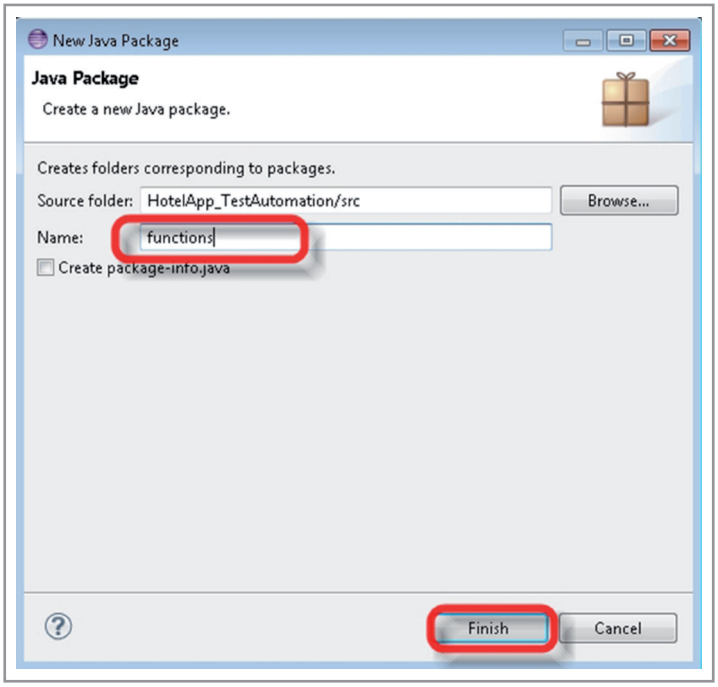


Figure 14.2 – Package Name

A new package is created with the name **functions**.

3. Right click on package **functions** and select **New → Class**

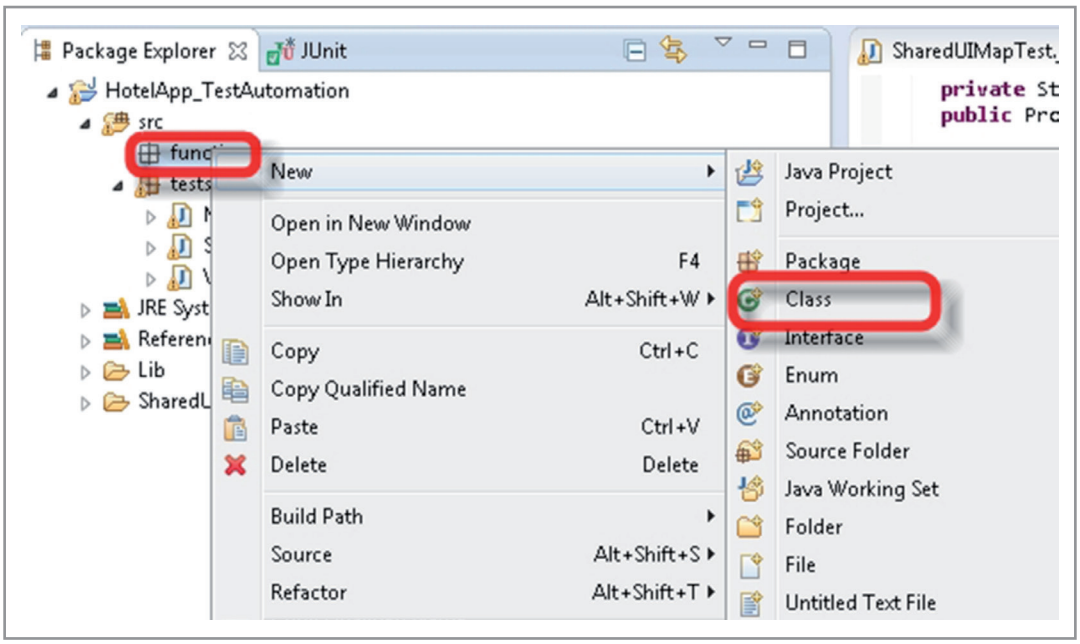


Figure 14.3 – New Class Creation

22

Exception Handling in WebDriver

We have already learned about exceptions and their handling techniques in the chapter on Basics of Java. An *exception* is an event which occurs during the execution of a program that disrupts the normal flow of the program's instructions.

Exception handling in Selenium is also a crucial exercise. Most of the time when a selenium script fails, it is because it has landed into an exception. The cause could be anything like:

- Element not found
- Couldn't click the element
- Element not visible

The moment the driver comes across an exception it will halt the test. So it's important for a tester to foresee these exception conditions and handle them according to the script or test requirements. This way the script failures are contributed to failures of test conditions and not to unhandled code exceptions. So, we have a bug corresponding to every test failure- which is our ultimate goal.

To catch an exception we first put the code which we suspect will throw an error into a **try** block like

```
WebElement txtbox_username = driver.findElement(By.id("username"));
try{
if(txtbox_username.isEnabled()){
txtbox_username.sendKeys("adactin123");}
 }
 catch(NoSuchElementException e){
System.out.println(e.toString());}
```

This is followed by a **catch block** of code where we tell the system what should be done when the exception occurs. Generally this is where we display the message of the exception object so that we know which exception has occurred and why.

22.1 Handling WebDriver Exceptions

In WebDriver we can use **try-catch** blocks or the **throws** statement with the purpose of handling the exceptions. The key point is that the exceptions we are catching here are Selenium exceptions rather than Java exceptions.

Test Scenario – If we provide an invalid username and password to the Login function, then the script should exit gracefully with a message.

Let us follow the steps to implement the above scenario.

1. Now what visual cue tells us that a user is logged in to the application? There can be multiple visual cues but let us pick one of them being a welcome message.

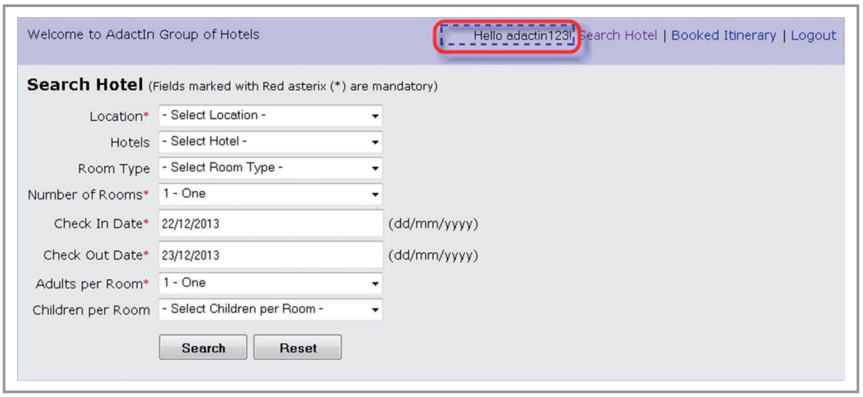


Figure 22.1 – User Welcome message

2. Let's use Firebug/FirePath to get its locator value

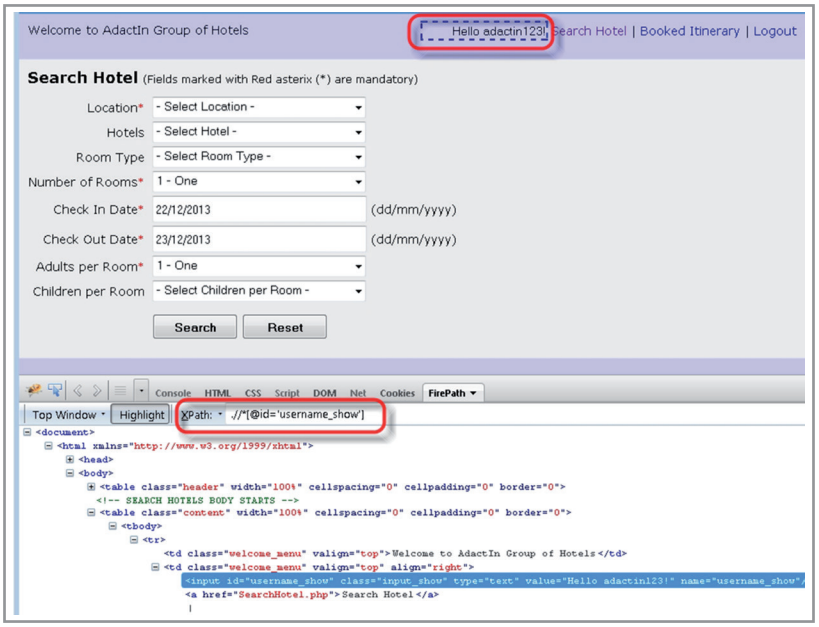


Figure 22.2 – Locator value for Welcome message

You see in the above snapshot the value for locator id is **username_show**

3. Add this to our SharedUIMap.properties file for further use

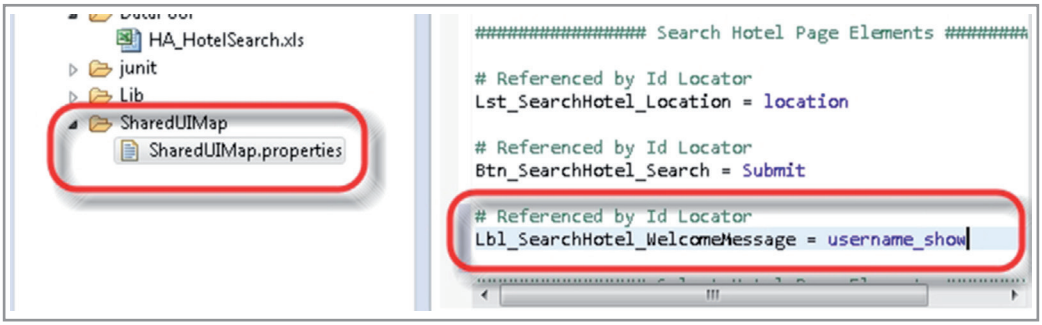


Figure 22.3 – Welcome message added to Shared UI Map

4. Double click and open our HotelApp_BusinessFunctions.java script to view our existing login function

Figure 22.4 – Existing Login Function

5. Modify the function as given below to handle a successful or unsuccessful login

```
public void HA_BF_Login (WebDriver driver, String sUserName, String sPassword ) throws
InterruptedException{



// Provide user name.

                driver.findElement(By.xpath(prop.getProperty(“Txt_Login_Username”))).
clear();

                driver.findElement(By.xpath(prop.getProperty(“Txt_Login_Username”))).
sendKeys(sUserName);


 // Provide Password.

driver.findElement(By.id(prop.getProperty(“Txt_Login_Password”))).clear();

        driver.findElement(By.id(prop.getProperty(“Txt_Login_Password”))).
```

23

Reporting in Selenium

One of the very important features of a test automation solution is its reporting structure. After test execution we inspect the test report for results and defect detection. Selenium does not have its own mechanism for reporting results. Rather, it allows the automation tester to build their own reporting structure, customized to their needs, using features of the programming language of your choice.

As part of this section, we are going to try to understand Test Framework Reporting tools

Key objectives:

- Test Framework Reporting Tools
- Configuring Junit HTML Report
- Configuring TestNG reports
- Custom Excel or Database reports

23.1 Test Framework Reporting Tools

Building your own reporting structure! It's great! But what if you simply want something quick that's already done for you? Often an existing library or test framework can meet your needs faster than developing your own test reporting code.

Test frameworks are available with all programming languages. Along with their primary function of providing a flexible test engine for executing your tests, they also include library code for reporting results. For example, Java has two commonly used test frameworks, JUnit and TestNG. .NET also has its own, NUnit.

What's The Best Approach?

Most people new to testing frameworks will begin with the framework's built-in reporting features since that's less time consuming than developing your own.

As you begin to use Selenium no doubt you will start putting in your own "print statements" for reporting progress. That may gradually lead to you developing your own reporting, possibly in parallel to using a library or test framework. Regardless, after the initial, but short learning curve, you will naturally develop what works best for your own situation, existing testing frameworks or your custom framework.

Test Reporting Examples in Java

We'll direct you to some specific tools supported by Selenium. The ones listed here are common and have been used extensively (and therefore recommended).

- If Selenium Test cases are developed using JUnit then JUnit Report can be used to generate test reports. To use JUnit Report you would need to integrate Eclipse with ANT.
- If Selenium Test cases are developed using TestNG then no external task is required to generate test reports. The TestNG framework generates an HTML report which list details of tests.

Advantages of using these frameworks:

- Very good reporting structure is available
- Can generate XML, HTML reports
- There are options available to create test methods, test suites, etc.
- Utilizes Selenium IDE or Firebug/FirePath to record test scripts

Disadvantages:

- We will not be able to define our own reporting format

23.2 Configuring JUnit HTML Reports

To create a JUnit based HTML report we will be using ANT with Eclipse. We had installed WinANT as part of the earlier setup components chapter.

ANT - Apache Ant is a Java based build tool from Apache whose aim is to drive processes described in build files as targets and extension points dependent upon each other. The main known usage of Ant is the build of Java applications. Ant supplies a number of built-in tasks allowing to compile, assemble, test and run Java applications. Ant can also be used effectively to build non Java applications, for instance C or C++ applications. More generally, Ant can be used to pilot any type of process which can be described in terms of targets and tasks.

Apache Ant's build files are written in XML and take advantage of the open standard, portable and easy to understand nature of XML.

Ant is extremely flexible and does not impose coding conventions or directory layouts to the Java projects which adopt it as a build tool.

Key steps in creation of JUnit HTML report include

- Generate ANT Build
- Run the Build file as Ant Build and Junit report option

25

Continuous Integration with Jenkins

Why do we need Continuous Integration tools for test automation?

Continuous Integration (CI) tools assists in creating builds frequently (usually on a daily basis) and running developer driven tests (unit tests) to provide timely feedback on application quality.

We can integrate our Selenium based functional test automation scripts with CI tools to execute our scripts as soon as a new build is created which will provide instant feedback on application issues.

Popular open source tools include Hudson, Jenkins (the fork of Hudson), CruiseControl and CruiseControl.NET.

Commercial tools include ThoughtWorks' Go, Urbancode's Anthill Pro, Jetbrains' Team City and Microsoft's Team Foundation Server.

As part of this chapter we will learn how Selenium WebDriver scripts integrate with Jenkins, one of the popular open source CI tools.

About Jenkins

Jenkins is a popular continuous integration server in the Java development community. It is derived from the Hudson CI server. It supports configuration management tools including CVS, Subversion, Git, Mercurial, Perforce, and ClearCase, and can execute Apache Ant and Apache Maven based projects as well as arbitrary shell scripts and Windows batch commands.

Jenkins can be deployed to set up an automated testing environment where you can runSelenium WebDriver tests unattended based on a defined schedule, or every time changes are submitted in configuration.

Key objectives:

- Install Jenkins tool
- Jenkins Configuration
- Run Jenkins with ANT
- Scheduling Auto-Runs

25.1 Installing Jenkins Tool

Let us first install Jenkins.

> Note: In a typical software development environment you would already have it installed by the development team.

1. Go to URL - http://jenkins-ci.org/
2. Download the Jenkins for the correct environment (in our case it is Windows)

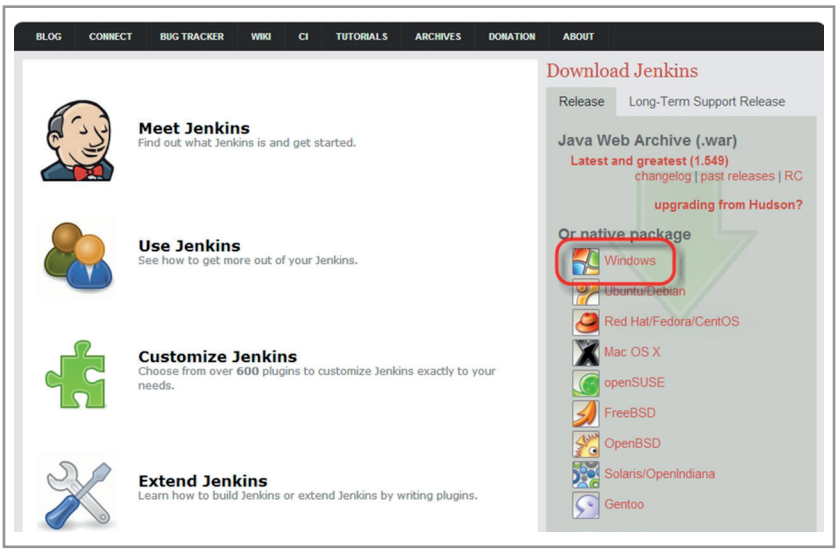
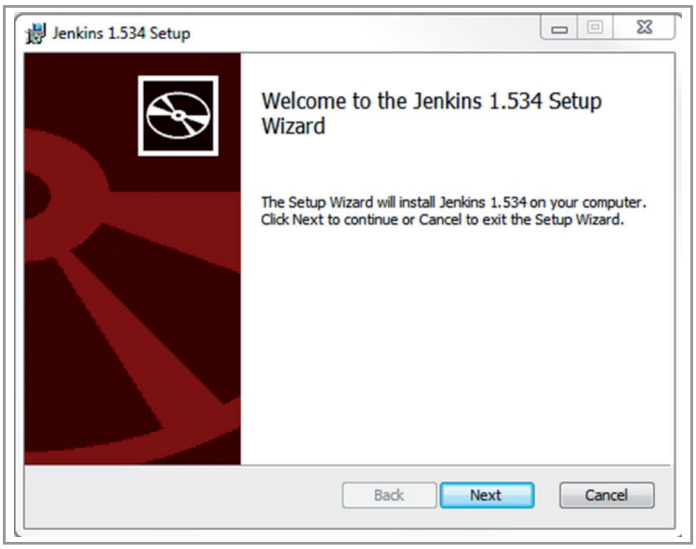


Figure 25.1- Download Jenkins

3. Unzip the install file and click on Setup.exe (incase of windows)



27

Selenium Functions, Common Questions and Tips

In this chapter we will try to address a few of the important selenium functions and other common questions and tips that can be used in Selenium.

Key objectives:

- How to use JavaScript?
- How to take a Screen Shot?
- How to use Keyboard or Mouse movements?
- How to read row, columns and cells data from a table?
- Working with multiple browsers
- How to maximize the Browser window
- Checking an Element's Presence
- Checking an Element's Status
- Working with drop-down lists
- Working with Radio buttons and groups
- Working with Checkboxes
- Measuring Response time for performance testing using timer
- Xpath and Properties finder in IE and Chrome browsers
- How to use WebDriver test remotely using Selenium Grid?

27.1 How to use JavaScript

Selenium WebDriver API provides the ability to execute JavaScript code with the browser window. This is a very useful feature where tests need to interact with the page using JavaScript. Using this API, client-side JavaScript code can also be tested using Selenium WebDriver.

For those browsers that support it, you can execute JavaScript by casting the WebDriver instance to a JavascriptExecutor .

Example

Below code executes javascript and returns the Web page title

```
JavascriptExecutor js = (JavascriptExecutor) driver;

// returns Web page title

String title = (String) js.executeScript("return document.title");

//returns handle to Webelement with id myid

WebElement element = (WebElement)js.executeScript("return document.
getElementById(myid)");
```

Figure 27.1 – Using JavaScript Executor

You need to return from your Javascript snippet to return a value, so:

js.executeScript("document.title");

will return null, but:

js.executeScript("return document.title");

will return the title of the document.

> **Note:** Based on the type of return value, we need to cast the executeScript() method. For decimal values, Double can be used, for non-decimal numeric values Long can be used, and for Boolean values Boolean can be used.

Example

Below code will return the count of combo boxes on the Search Hotel Page