Rex Jones II, CSTE, TMap

PART 1
ABSOLUTE BEGINNER

Selenium WebDriver for Functional Automation Testing

Your Beginners Guide

FREE BOOK

Preface

Software QA/Testers are expected to test more development code in less time. The increased production from developers can cause testers to become bottlenecks. However, testing a web application utilizing automation helps speed up the delivery of software products. Selenium WebDriver reduces the time it takes to execute a test. In addition, test automation controls the execution of a test, verifies whether actual results return expected results and report results (Pass/Fail). The purpose of this book is to provide building blocks for automating any web application using Selenium WebDriver.

Target Audience

Absolute Beginner

Why You Should Learn Selenium WebDriver?

Selenium WebDriver is a powerful open source test automation effort. Open source refers to free software that is developed for the community. Many test automation tools such as Selenium IDE provide record and playback features. Record and playback allow users to record their actions and replay the actions any number of times. However, Selenium WebDriver does not include a record feature but allow an automation engineer to execute Test Scripts any number of times.

Selenium WebDriver Test Scripts are created and executed within an Integrated Development Environment (IDE). The IDE provide standards, assist with project management, and organize Test Scripts in a structure manner. Employing Selenium WebDriver is a valuable automation effort for assuring quality.

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

About the Author

Rex Allen Jones II is a QA/Software Tester with a passion for sharing knowledge about testing software. He has been watching webinars, attending seminars, and testing applications over 10 years. Mr. Jones graduated from DeVry University with a Bachelor's of Science degree in Computer Information Systems (CIS).

Rex is an author, consultant, and former Board of Director for User Group: Dallas / Fort Worth Mercury User Group (DFWMUG) and member of User Group: Dallas / Fort Worth Quality Assurance Association (DFWQAA). In addition to his User Group memberships, he is a Certified Software Tester Engineer (CSTE) and has a Test Management Approach (TMap) certification.

Mr. Jones' advice for people interested in Automation Testing is to learn the programming language. This advice led him to write 4 programming books "(Part 1 & Part 2) You Must Learn VBScript for QTP/UFT" and "(Part 1 & Part 2) Java 4 Selenium WebDriver". VBScript is the programming language for Unified Functional Testing (UFT) formerly known as Quick Test Professional (QTP) and Java is one of the programming languages for Selenium WebDriver.

3 Tips To Master Selenium Within 30 Days
http://tinyurl.com/3-Tips-For-Selenium

Free Webinars, Videos, and Live Trainings
http://tinyurl.com/Free-QTP-UFT-Selenium

In addition to the 4 programming books, Mr. Jones wrote 2 more books. The 5th book is named Absolute Beginner (Part 1) Selenium WebDriver for Functional Automation Testing which provides a deep foundation of Selenium WebDriver. Finally, a 6th book named Getting Started With TestNG (A Java Test Framework). All books are available in Paperback, eBook, and PDF.

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Rex Jones' Contact Information

Email Address: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34
Books: http://tinyurl.com/Rex-Allen-Jones-Books
Twitter: @RexJonesII
Skype: rex.jones34

Free Webinars, Videos, and Live Training

Mr. Jones plans to have **free** step-by-step demonstration webinars, videos, and live trainings walking people through concepts of Selenium and QTP/UFT from A - Z. The material will teach/train individuals the fundamentals of the programming language, fundamentals of Selenium and QTP/UFT, and important concepts of Selenium and QTP/UFT. All of the webinars, videos, and live training will be directed toward beginners as well as mid-level automation engineers.

Sign Up to Receive

1. 3 Tips To Master Selenium Within 30 Days
   http://tinyurl.com/3-Tips-For-Selenium

2. 3 Tips To Master QTP/UFT Within 30 Days
   http://tinyurl.com/3-Tips-For-QTP-UFT

3. Free Webinars, Videos, and Live Trainings
   http://tinyurl.com/Free-QTP-UFT-Selenium

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Acknowledgements

I would like to express my gratitude to my wife Tiffany, children Olivia Rexe' and Rex III, family, friends, and the many people who provided encouragement. Writing this book took time and your support helped pushed this book forward.

Thank You,

Rex Allen Jones II

Copyright, Legal Notice, and Disclaimer

Please note that much of this publication is based on personal experience and anecdotal evidence. The author has made every reasonable attempt to produce accurate content in this book. He assumes no responsibility for unknown errors or omissions. Therefore, the purchaser should use this information as he/she sees fit.

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Table of Contents

Table of Contents

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Chapter 1
Introduction to Selenium WebDriver

History

According to SeleniumHQ, the history of Selenium starts in 2004 at Thoughtworks with Jason Huggins building the core mode as JavaScriptTestRunner. He built the core mode because he did not want to manually step through the same test after every change. Therefore, Jason developed a JavaScript library that permitted him to run test repeatedly against multiple browsers such as Firefox, Google, and Internet Explorer.

Soon after, fellow coworker Paul Hammant saw a demo by Jason and started discussions about making Selenium open source. Open source is free software developed by the community and for the community. The JavaScript library became the core of Selenium that underlines Selenium Remote Control (RC).

Selenium RC is a powerful automation tool that allows an engineer to control the browser. The tool drastically improves productivity by reducing test time and cost. Although powerful, Selenium RC has drawbacks due to its JavaScript based automation engine. The drawback makes certain tasks impossible to carry out.

As a result of the drawbacks, an engineer named Simon Stewart started working on a project in 2006 called WebDriver. The goal of WebDriver was to address Selenium's drawbacks. After WebDriver solved the restrictions of Selenium, both developers decided to merge the two projects. The merge made Selenium WebDriver (also known as Selenium 2) a very robust test automation effort.

Selenium WebDriver released in 2011 and became the successor to Selenium RC. Selenium WebDriver runs on multiple platforms and supported by multiple programming languages. The

platforms are Windows, Macintosh, and Linux while the programming languages are C # pronounced as C Sharp, Java, Python, and Ruby.

Chapter 1 provides an overview of this book and introduce the following:

- ✓ WebDriver and WebElements
- ✓ Selenium WebDriver Locator Types
- ✓ Selenium WebDriver Method Categories
- ✓ Installations

WebDriver and WebElements

Most software applications are web-based applications designed to run on browsers. WebDriver is an interface utilized for testing web applications that contain WebElements (see *WebDriver and WebElements in Chapter 2*). WebElements (also known as elements) are buttons, text boxes, checkboxes, drop-down menus, hyperlinks, etc. via a web application. Finding an element then performing an action on the element are keys to automating all applications.

Selenium WebDriver Locator Types

Selenium WebDriver has 8 locator types to assist with finding an element. The 8 locators in alphabetical order are className, cssSelector, id, linkText, name, partialLinkText, tagName, and xpath. Each locator requires a value to find the element. Most of the values can be inspected from a web application's HyperText Markup Language (HTML). Other times, to find an exact match, the value must be taken directly from the application. The locators are explained in the following chapters:

- Chapter 3 – ID
- Chapter 4 – Name
- Chapter 5 – XPath
- Chapter 6 – CSS Selector
- Chapter 7 – Link Text

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Selenium WebDriver Method Categories

There are 5 Method Categories for Selenium WebDriver. A method is a block of code that performs an action / task. Some methods within the same category can perform a similar task. For example, close() and quit() are methods within the Browser Category. Both methods close the browser's window. However, the close method closes the current window while quit closes every associated window.

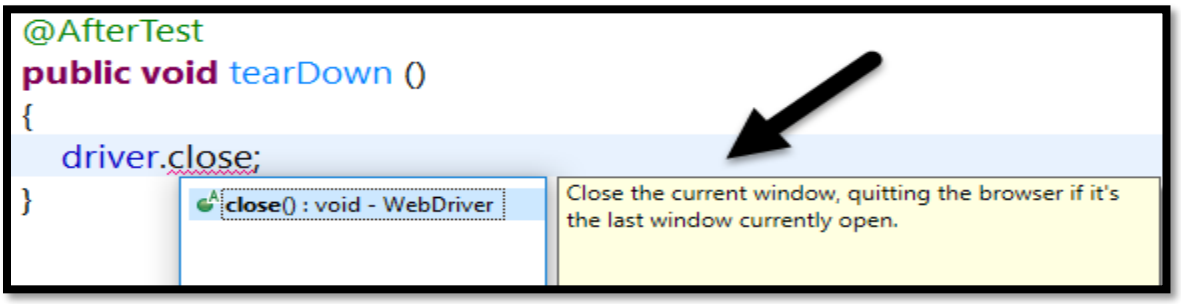


Figure 1.1 – Close Method Closes The Current Window

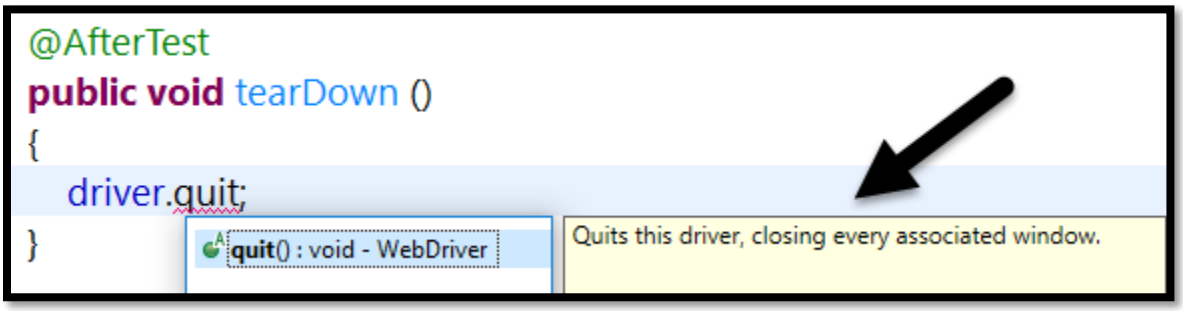


Figure 1.2 – Quit Method Closes Every Associated Window

Also, methods in different categories can performs the same task. The Browser Category has a method called get() that loads a new web page while Navigation Category has a method called navigate().to() that loads a new web page.

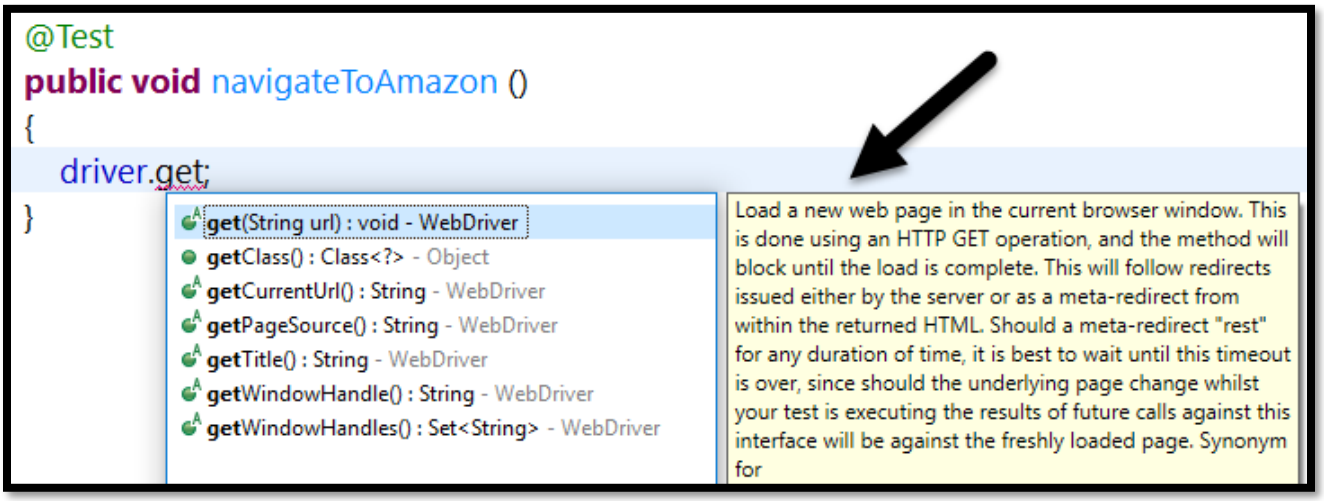


Figure 1.3 – Get Method Loads A New Web Page

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

```
@Test
public void navigateToAmazon ()
{
   driver.get("https://www.amazon.com/");
}
```

Figure 1.4 – Load Amazon Web Page via Get Method

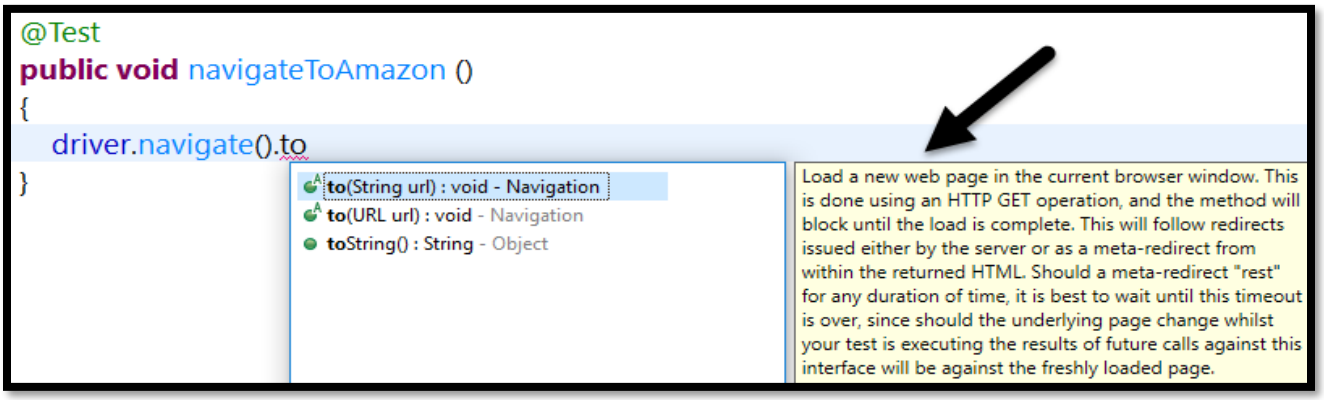


Figure 1.5 – Navigate.To Method Loads A New Web Page

```
@Test
public void navigateToAmazon ()
{
   driver.navigate().to("https://www.amazon.com/");
}
```

Figure 1.6 - Load Amazon Web Page via Navigate.To Method

All 5 Method Categories are explained in the following chapters:

- Chapter 11 – Browser Methods
- Chapter 12 – Wait Methods
- Chapter 13 – Navigation Methods
- Chapter 14 –WebElement Methods
- Chapter 15 – Switch Methods

Installations

Eclipse requires several installations to execute automation Test Scripts. It is best to provide all of the installation steps in one section rather than listing the steps in different parts of the book. The following is a list of installations to get started with Selenium WebDriver using Java:

- Verify The System Type
- Install Browsers
- Install Java Development Kit (JDK)
- Install Eclipse IDE
- Install TestNG
- Configure WebDriver
- Download Browser Drivers
- Download Notepad ++

Note: Steps for installing Selenium IDE, Firebug, and FirePath has been removed since Firefox no longer supports those plugins.

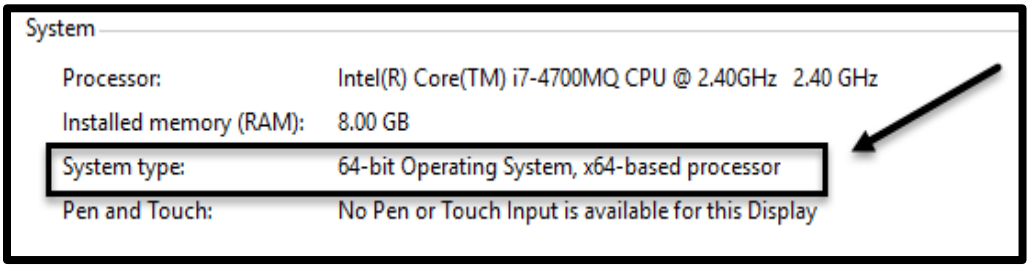
Verify The System Type

A verification of the System Type determines the computer's version. Knowledge of the System Type helps when downloading/installing a particular product. All types of errors such as performance issues can occur soon after installing the wrong System Type. The following are steps to verify a computer's System Type:

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Steps To Verify The System Type:

1. Go to Start menu
2. Select System
3. System type = "i.e., 64-bit Operating System - Windows 64"



Install Browsers

A browser is a software application used to access the internet. It provides a way to interact with all of the information on the World Wide Web. There are many browser types that contain similar functionalities. Each browser retrieves and presents information via hyperlinks. The most commonly used browsers are Firefox, Google Chrome, and Internet Explorer (IE). An installation of at least one browser is required in order to install the other products. The following are steps to install Firefox, Google Chrome, and Internet Explorer:

- Steps To Install Firefox
- Steps To Install Google Chrome
- Steps To Install Internet Explorer

Steps To Install Firefox:

1. Go to https://www.mozilla.org/en-US/firefox/new/
2. Click Free Download
3. Click Save then save to a location
4. Click Run

5. Click Yes
6. Click Next
7. Click Install
8. Click Finish

Steps To Install Google Chrome:

1. Go to https://support.google.com/chrome/answer/95346?hl=en
2. Click Download
3. Click Accept and Install
4. Click Run
5. Click Yes

Steps To Install Internet Explorer:

1. Go to https://www.microsoft.com/en-us/download/details.aspx?id=39232
2. Click Download
3. Click Run
4. Click Continue
5. Click Install

Note: The steps to install each browser are subject to change.

Install Java Development Kit (JDK)

The Java Development Kit (JDK) is a software development environment used for writing code in Java. This book presumes the reader has knowledge of Java. If not, (Part 1 and Part 2) Java 4 Selenium WebDriver covers core Java which is important for Selenium WebDriver. JDK includes many required components for creating and testing applications. Some of the components are Java Runtime Environment (JRE), Java Compiler, Java Interpreter, and Java Archiver (JAR).

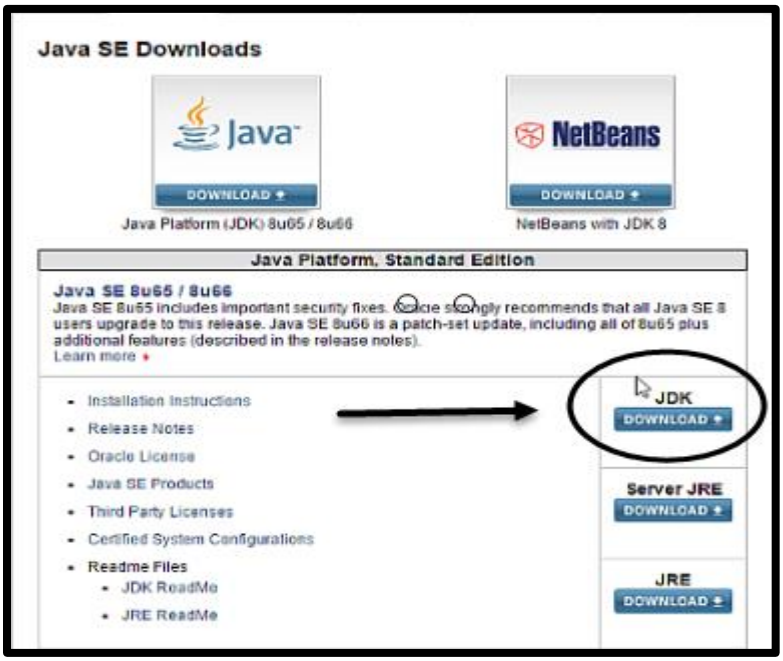
Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

- Java Runtime Environment (JRE) – provides the requirements to execute code in a web browser
- Java Compiler – primary program that reads class definitions then compiles it into bytecode class files
- Java Interpreter – primary program that executes bytecode for Java Virtual Machine
- Java Archiver (JAR) – files used to combine Java class files

The following are steps to install JDK:

Steps To Install JDK:

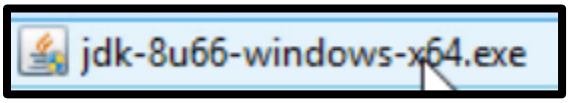
1. Go to Java SE Downloads
   http://www.oracle.com/technetwork/java/javase/downloads/index.html
2. Click the JDK Download button



3. Click Accept License Agreement in the Java SE Development Kit 8u66 section
   Note: There may be a more recent version than 8u66

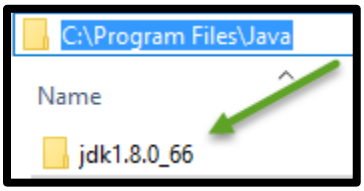
Product / File Description	File Size	Download
Linux x86	154.67 MB	jdk-8u66-linux-i586.rpm
Linux x86	174.83 MB	jdk-8u66-linux-i586.tar.gz
Linux x64	152.69 MB	jdk-8u66-linux-x64.rpm
Linux x64	172.89 MB	jdk-8u66-linux-x64.tar.gz
Mac OS X x64	227.12 MB	jdk-8u66-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.65 MB	jdk-8u66-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.05 MB	jdk-8u66-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	140 MB	jdk-8u66-solaris-x64.tar.Z
Solaris x64	96.2 MB	jdk-8u66-solaris-x64.tar.gz
Windows x86	181.33 MB	jdk-8u66-windows-i586.exe
Windows x64	186.65 MB	jdk-8u66-windows-x64.exe

4. Click the Download link for the appropriate System Type “i.e., Windows x64”
5. Go to the Download folder
6. Open the downloaded executable file



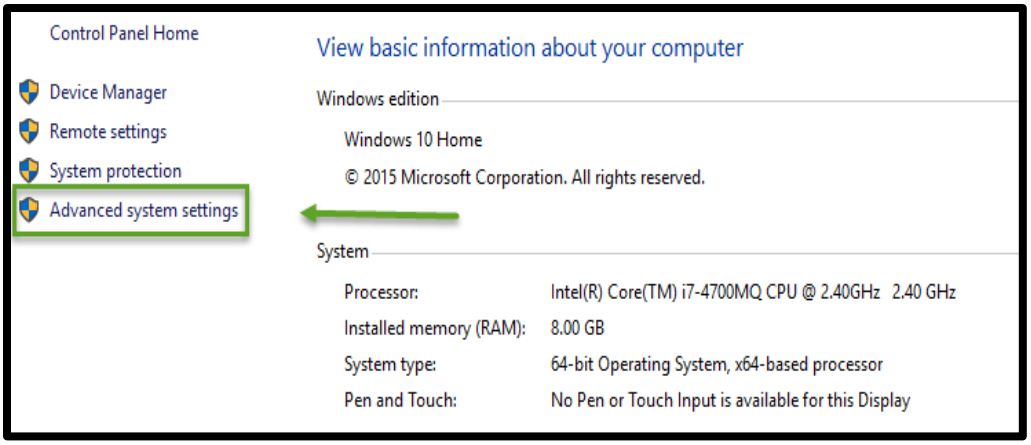
7. Click the Next button to Set Up Java SE Development Kit
8. Click the Next button for Custom Set Up
9. Click the Next button to Install to a specific location
   “i.e., C:\Program Files\Java”

10. Go to the location and Open the jdk folder “i.e., jdk1.8.0_66”

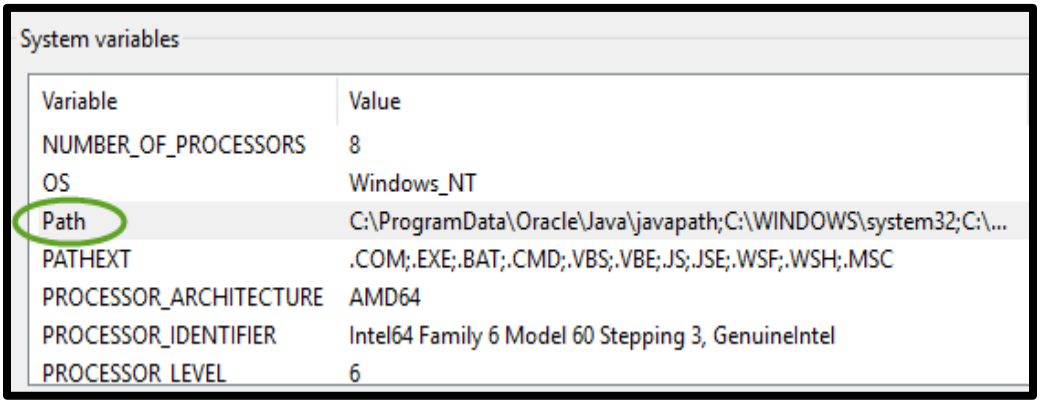


11. Open the bin folder

12. Copy the bin folder's location "i.e., C:\Program Files\Java\jdk1.8.0_66\bin"
13. Access the Advanced system settings via System

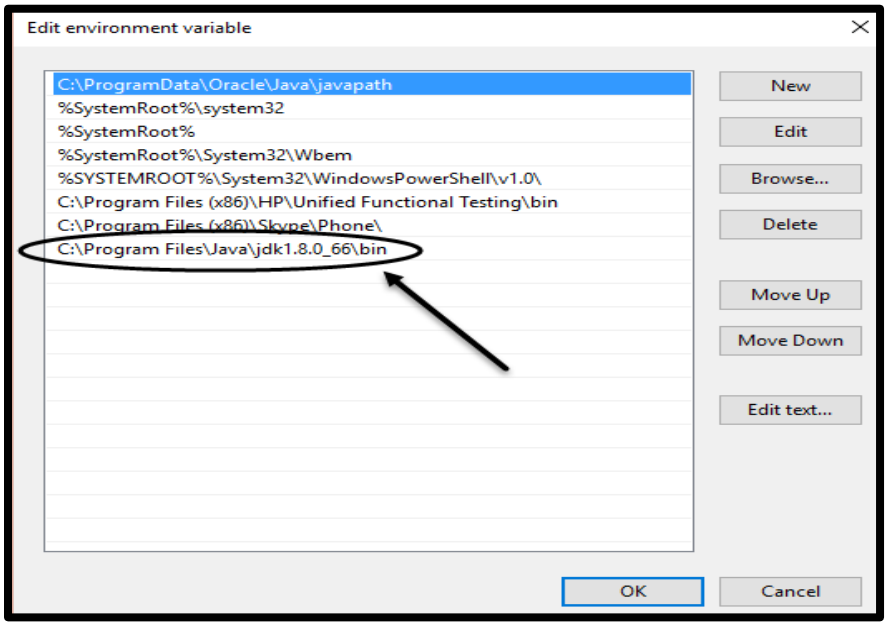


14. Click the Advanced tab
15. Click Environment Variables
16. Go to Path within System variables section



17. Click Edit

18. Paste the bin folder's location "i.e., C:\Program Files\Java\jdk1.8.0_66\bin"



19. Click OK
20. Click Apply
21. Click OK

Note: Steps 10 – 21 are optional but beneficial. Eclipse automatically searches for the path "i.e., C:\Program Files\Java\jdk1.8.0_66\bin" that is placed in the Environment Variables modal.

Install Eclipse IDE

Eclipse is an open source IDE used for developing and testing applications. An IDE is comprehensive whereby it contains many features. The source code editor and debugger are some of the features. A source code editor allows code creation while a debugger examines the created code. Eclipse supports multiple programming languages but mainly used for Java. One of the benefits of Eclipse is the use of plugins. The plugins allow customizations and additional functionalities. The following are steps to install Eclipse IDE:

Steps To Install Eclipse:

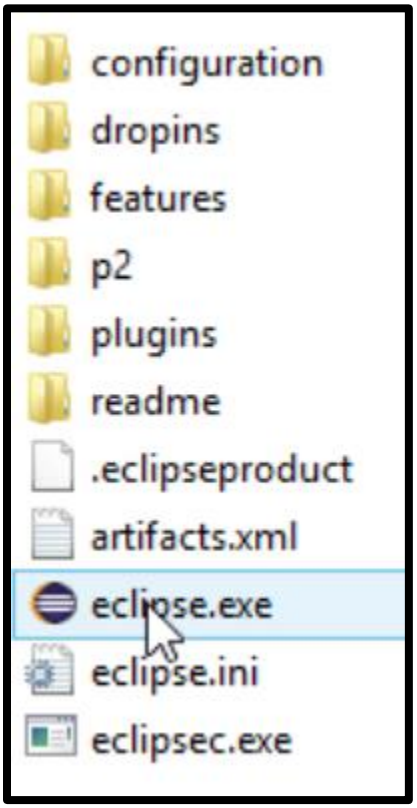
1. Go to https://eclipse.org/downloads/
2. Select the platform (Windows, Mac OS, or Linux)
3. Click the System Type "i.e., 64 bit" for Eclipse IDE for Java EE Developers

4. Choose a mirror close to you "i.e., Columbia University"

5. Go to Download folder

6. Right click the Eclipse zip file and Extract All files

7. Open the eclipse.exe file to launch Eclipse IDE
    a. Go the extracted folder “i.e., eclipse-jee-mars-1-win32-x86_64”
    b. Open eclipse folder
    c. Right click eclipse.exe and Select Open



8. Load Eclipse IDE

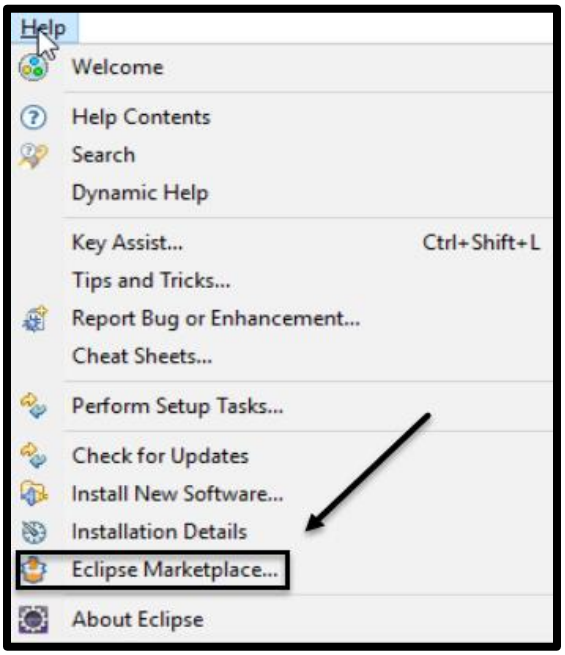
Install TestNG

TestNG is a powerful testing framework structured for Java. The framework is designed to cover all types of testing such as unit, regression, functional, end-to-end, and integration. Eclipse requires a TestNG installation to utilize TestNG features such as annotations "@BeforeTest, @Test, and @AfterTest". The following are steps to install TestNG:

Steps To Install TestNG:

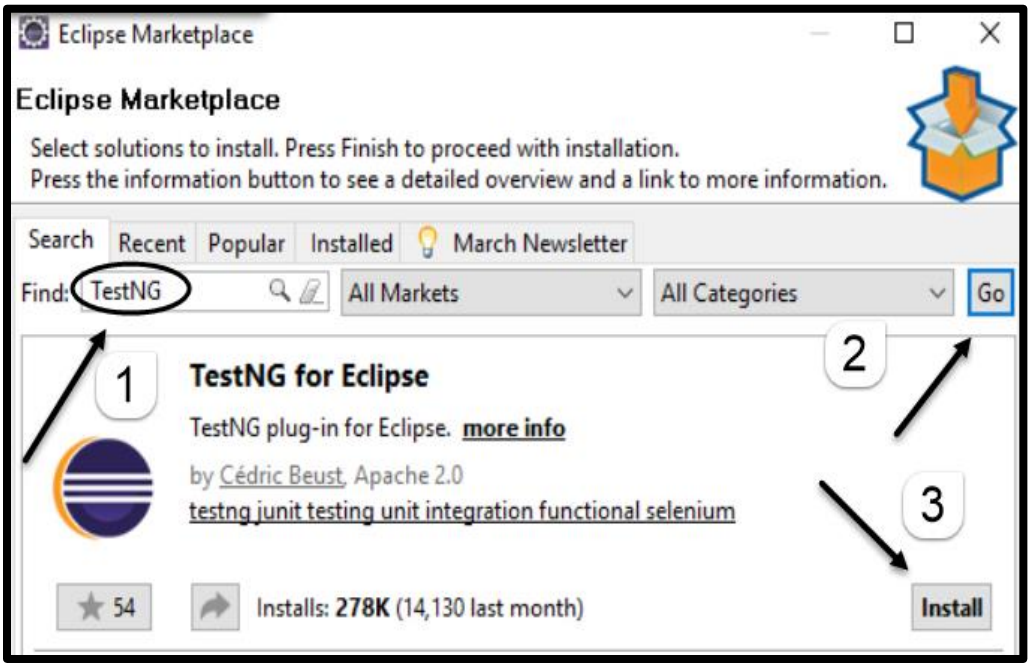
1. Open Eclipse
2. Load a Project "i.e., Hello World"

3. Click Help > Eclipse Marketplace

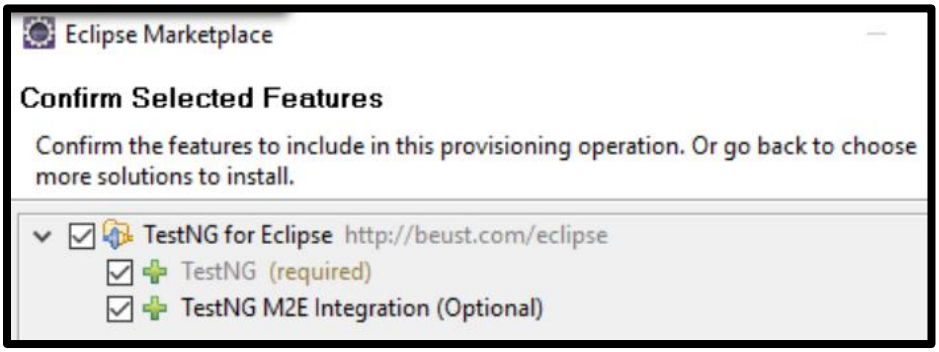


Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

4. Search for TestNG, Click the Go button, then Click the Install button

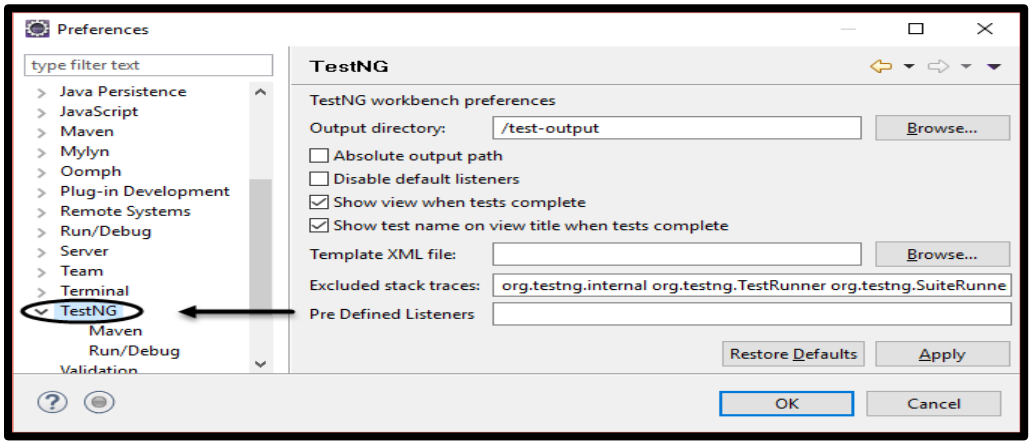


5. Click the Confirm button to confirm the following features:
   a. TestNG (required)
   b. TestNG M2E Integration (Optional)



6. Click the radio button “I accept the terms of the license agreement”
7. Click the Finish button

8. Click OK if a Security Warning appears in a modal
9. Click Yes to restart Eclipse
10. Verify TestNG has been added
    a. Re-open Eclipse after restart
    b. Click Window
    c. Click Preferences



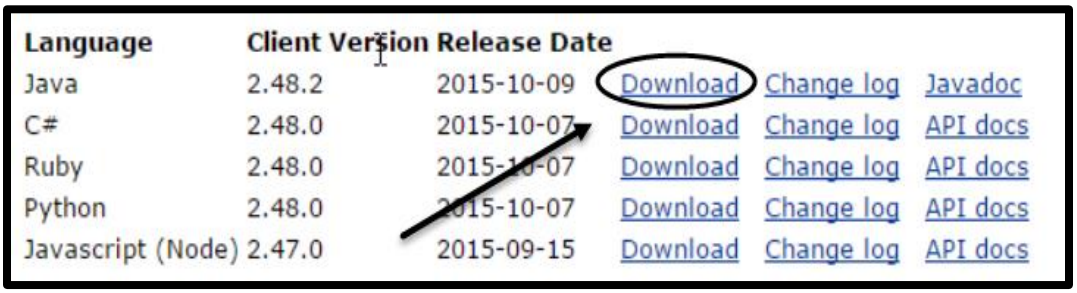
Configure WebDriver

As mentioned earlier, Selenium WebDriver supports multiple programming languages. The programming languages are C#, Java, Python, and Ruby. Download the client driver in order to execute Test Scripts for a particular language. In this case, WebDriver requires a client driver specific to Java. After downloading, configure the client driver in Eclipse by adding JAR files. A JAR file is a collection of multiple Java class files used to distribute Java libraries. Selenium HQ allows a download of the most recent client version for all supported languages. The following are steps to configure a Java client driver:

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

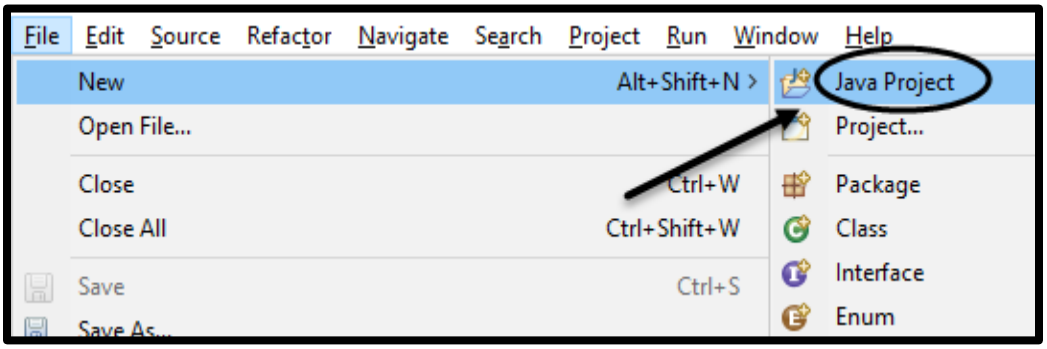
Chapter 1
Introduction to Selenium WebDriver (Part 1) Selenium WebDriver
Steps To Configure A Java Client Driver:

1. Go to http://docs.seleniumhq.org/download/
2. Download client driver for Java
   Note: As of version 3.13, the following steps are the same for Java.



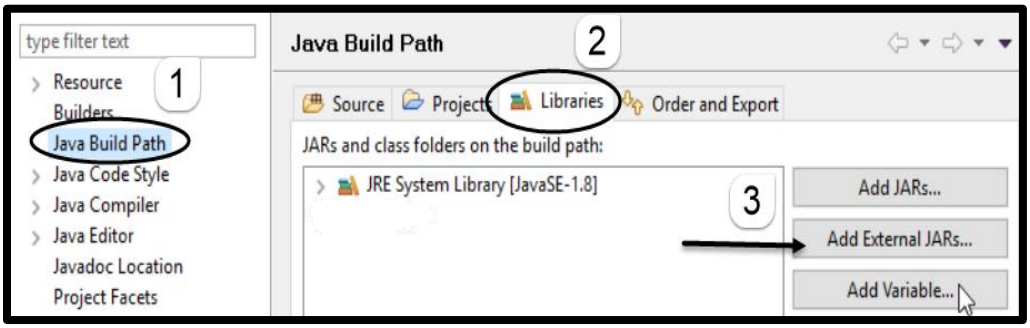
3. Go to Download folder
4. Right click the zip file "i.e., selenium-java-2.48.2.zip" and Extract All "jar files"

5. Open Eclipse IDE
6. Create New Project:
   Navigate to File > New > Java Project > Project Name "i.e., Hello World" > Finish

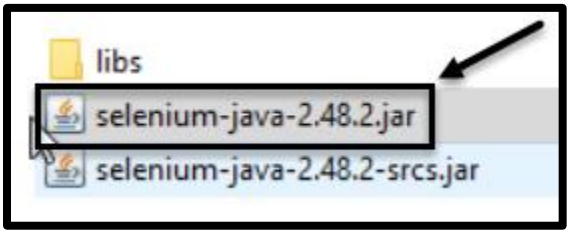


7. Click File > Properties

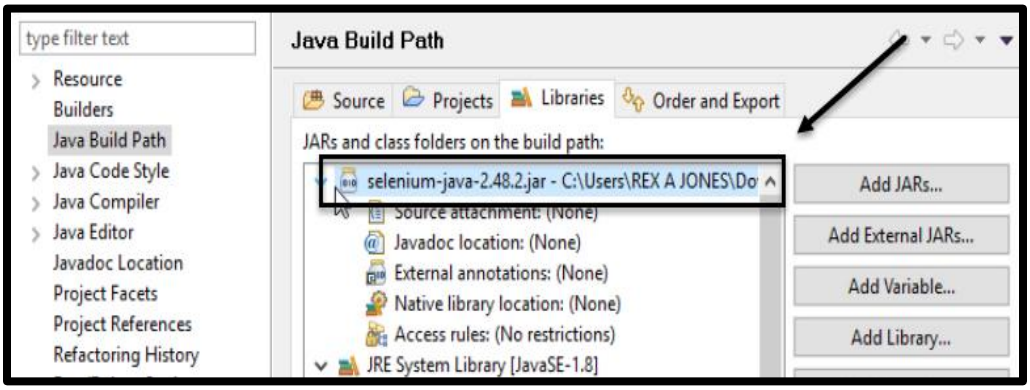
8. Select Java Build Path > Libraries > Add External JARs



9. Go to the Download folder and Select the extracted jar file "selenium-java-2.48.2.jar"



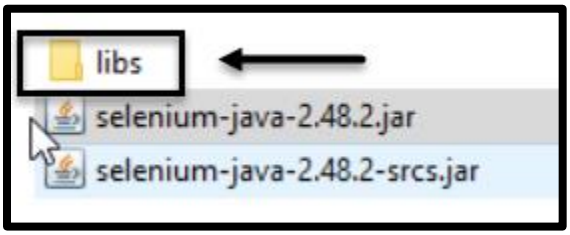
10. Click Open > jar file "i.e., selenium-java-2.48.2.jar" added to the Library



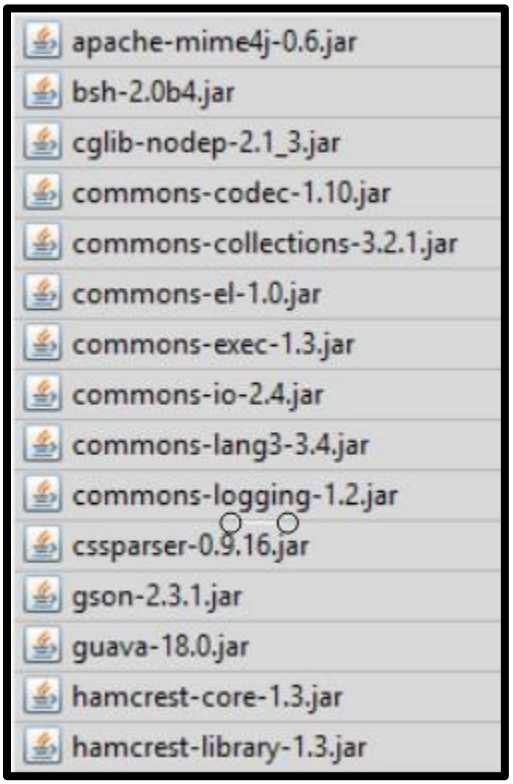
11. Click Add External JARs

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

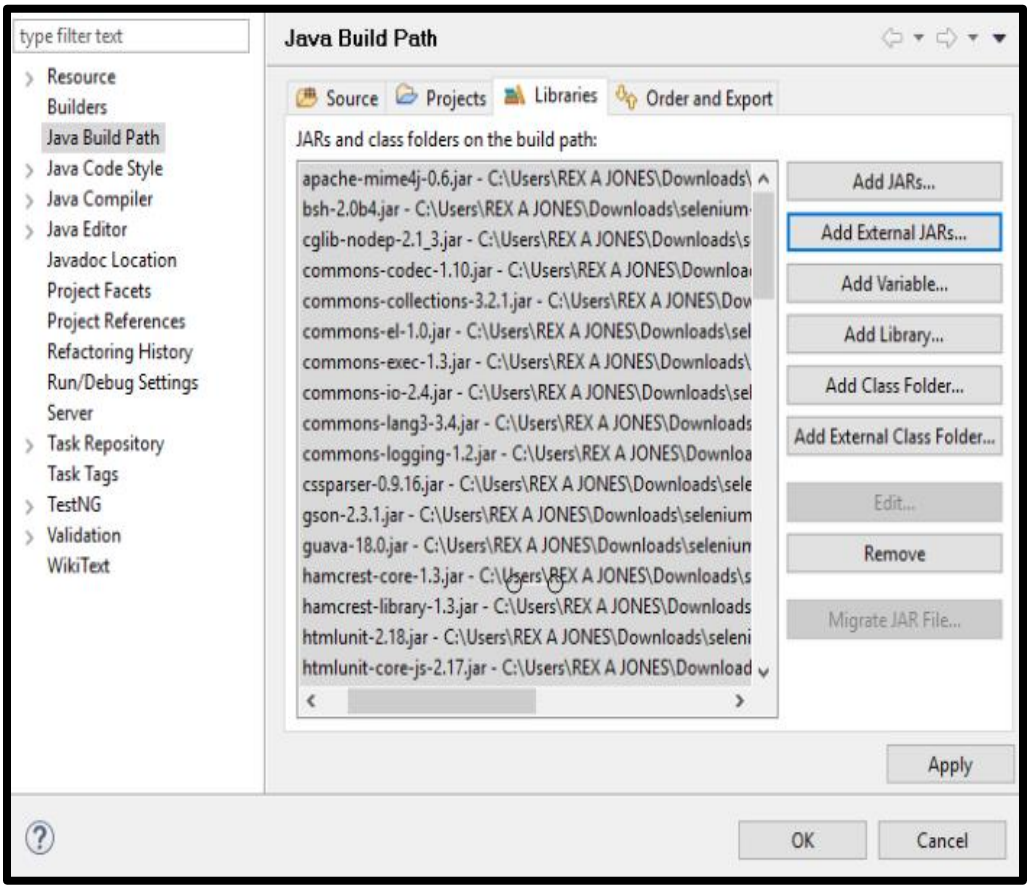
12. Select libs in the extracted folder "i.e., selenium-java-2.48.2"



13. Select all of the jar files in libs folder: Ctrl + A



14. All of the jar files are added to the Java Build Path Library > Click OK



Download Browser Drivers

According to Selenium HQ, "in order to create scripts that interact with the Selenium Server (Selenium RC, Selenium Remote WebDriver) or create local Selenium WebDriver scripts, you need to make use of language-specific client drivers". The core languages are C#, Java, JavaScript, Python, and Ruby. Most of the drivers execute on all major browsers. However, the following provides steps to download drivers for Google Chrome, Internet Explorer, and Firefox:

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

- Google Chrome
- Internet Explorer
- Firefox

Google Chrome

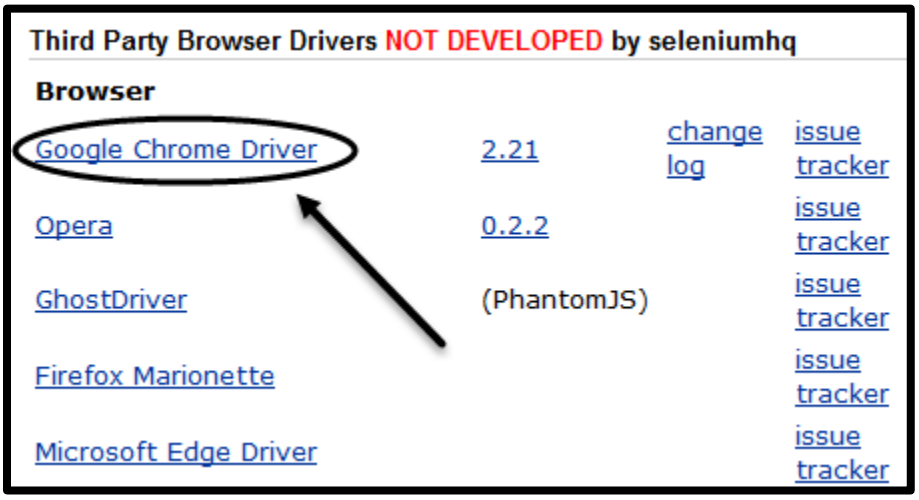
Google Chrome requires a driver called ChromeDriver to run Selenium WebDriver Test Scripts. WebDriver uses ChromeDriver to control testing performed on Google Chrome. An error occurs in Eclipse if there is not a driver download. However, the error provides the following link to download the latest ChromeDriver:

http://chromedriver.storage.googleapis.com/index.html

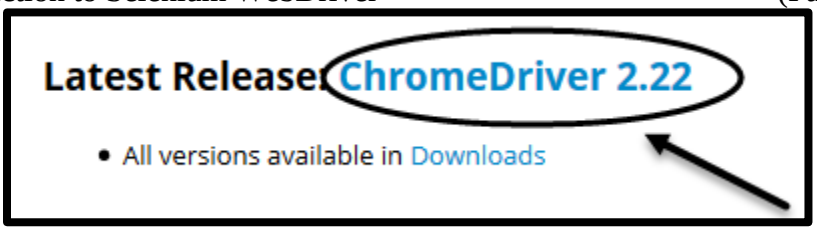
The following are steps to download ChromeDriver from Selenium HQ:

Steps To Download ChromeDriver:

1. Go to http://docs.seleniumhq.org/download/
2. Click the link for Google Chrome Driver in Third Party Browser Drivers section

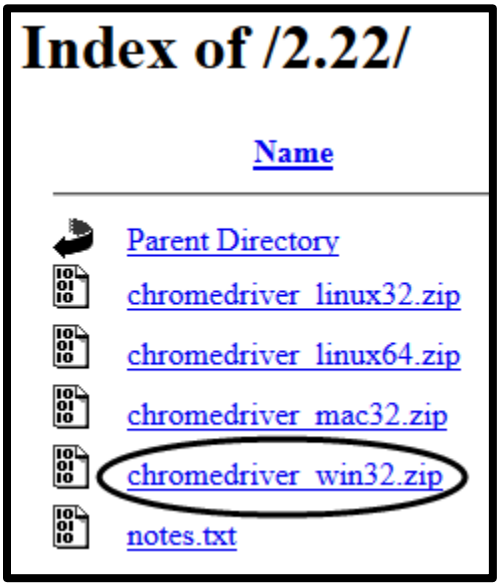


3. Click the link for the most recent release "i.e., 2.22" via https://sites.google.com/a/chromium.org/chromedriver/



4. Click the link for the most recent release again "i.e., 2.22" via https://sites.google.com/a/chromium.org/chromedriver/downloads

5. Download the system's zip file "i.e., chromedriver_win 32.zip"

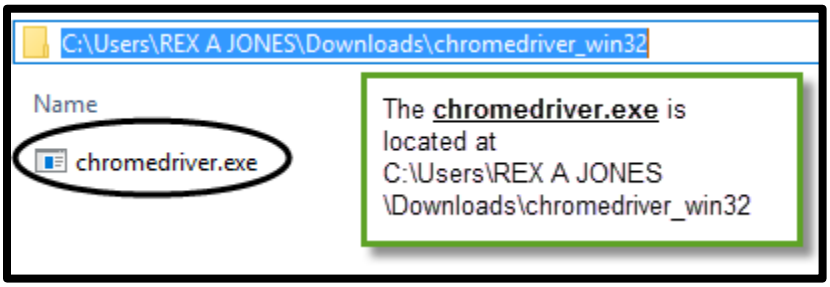


Note: Zip file for Windows 64 is not available. Therefore, the zip file for Windows 32 "chromedriver_win 32.zip" should be downloaded

6. Go to Download folder
7. Right click the zip file "i.e., chromedriver_win32.zip" and Extract All

8. Open the extracted folder "i.e., chromedriver_win32"

9. chromedriver.exe has been downloaded



Note: The chromedriver.exe is located at C:\Users\REX A JONES\Downloads\chromedriver_win32.

There are two ways to ensure Test Scripts run successfully:

- Add path to Environment Variables – System Variables
- Add path to Selenium WebDriver Test Script. The following is an example of using the chromedriver.exe in a Test Script:

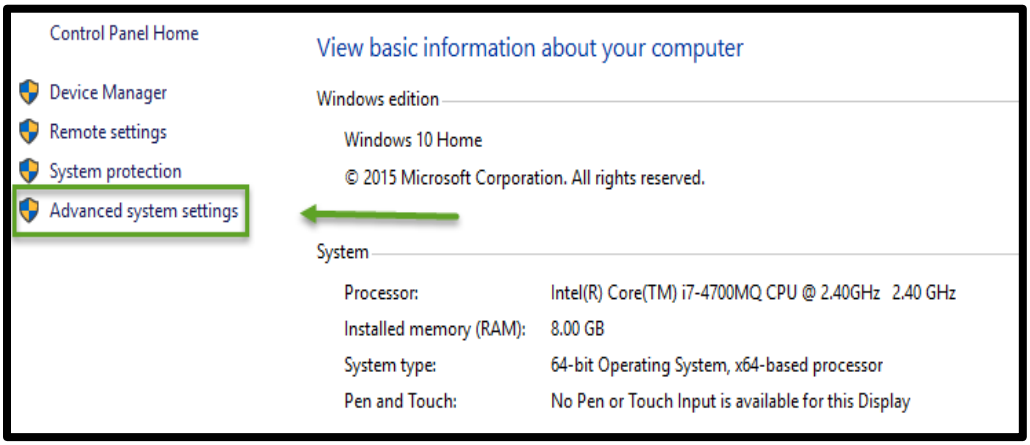
C:\\Users\\REX A JONES\\Downloads\\chromedriver_win32\\chromedriver.exe");

```
System.setProperty("webdriver.chrome.driver","C:\\Users\\REX A JONES\\Downloads\\chromedriver_win32\\chromedriver.exe");
```

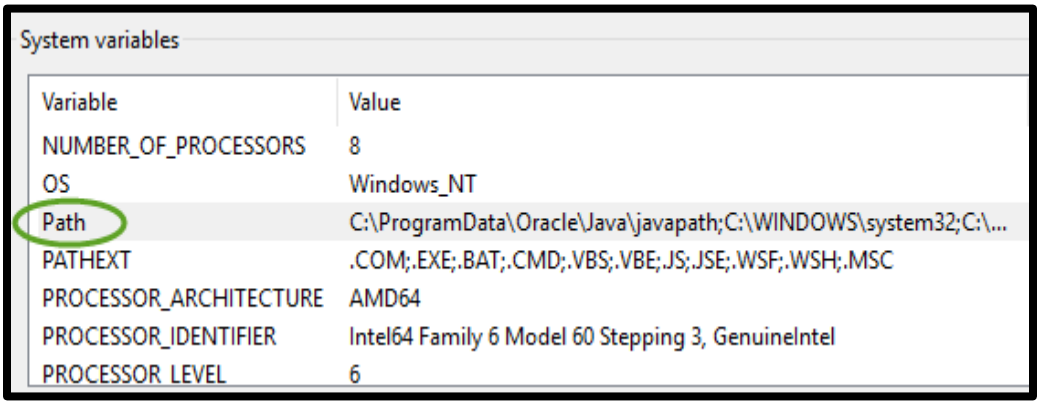
The following illustrates how to add the executable file “i.e., chromedriver.exe” path to Environment Variables - System Variables:

10. Copy the path “i.e., C:\Users\REX A JONES\Downloads\chromedriver_win32”

11. Access the Advanced system settings via System



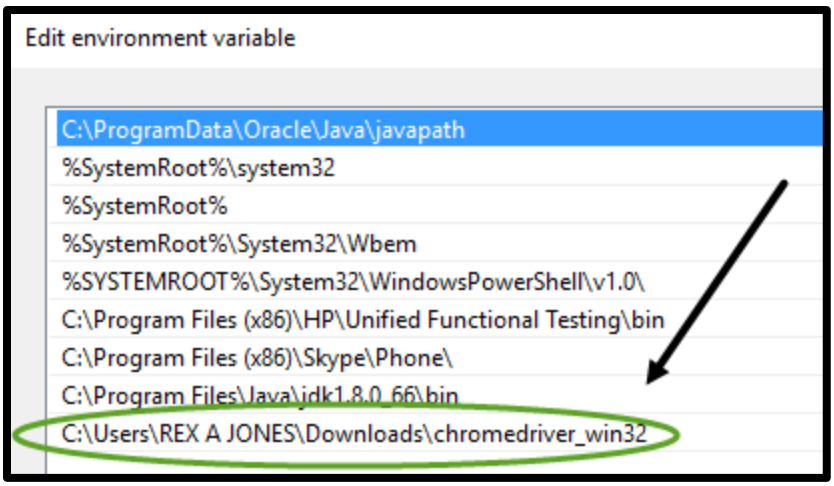
12. Click the Advanced tab
13. Click Environment Variables
14. Go to Path within System variables section



15. Click Edit

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

16. Paste the ChromeDriver's executable file location
    "i.e., C:\Users\REX A JONES\Downloads\chromedriver_win32"



17. Click OK
18. Click Apply
19. Click OK

Note: Most examples in this book include an extra code line via System.set*Property* when executing Test Scripts in Google Chrome. However, the preferred way is to add the path to Environment Variables – System Variables.

Internet Explorer

Internet Explorer requires a driver called IEDriverServer to run Selenium WebDriver Test Scripts. WebDriver uses IEDriverServer to control testing performed on Internet Explorer. An error occurs in Eclipse if there is not a driver download. Selenium HQ contains a section called Internet Explorer Driver Server that allows an IEDriverServer download. Follow the same steps (6 - 19) from Google Chrome's section after downloading IEDriverServer:

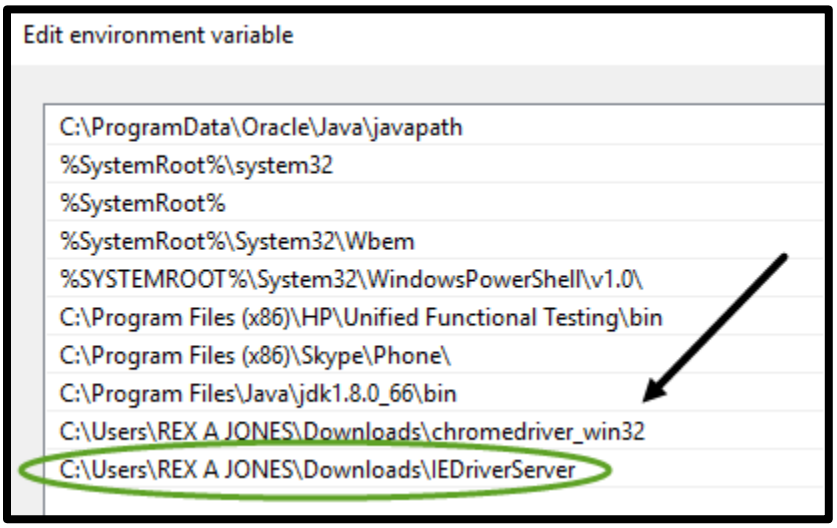
Figure 1.7 – Internet Explorer Driver Server

Eclipse provides a link to download IEDriverServer if an error occurs. The following are steps to download IEDriverServer directly from that link:

Steps To Download IEDriverServer:

1. Go to Internet Explorer Driver Server section on Selenium HQ page
2. Download the most recent Internet Explorer driver

3. Follow the same steps (6 - 19) from Google Chrome's section to execute Test Scripts for Internet Explorer browser.



Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Note: The security and Firewall exception must be set up / added before running Test Scripts in Internet Explorer.

4. Go to Tools in Internet Explorer
5. Select Internet Options
6. Select Security
7. Confirm the Security Level for all zones (Internet, Local Intranet, Trusted Sites and Restricted Access) are the same
    a. Uncheck or Check the checkbox for Enable Protected Mode
    b. Click Apply
    c. Click OK
    d. Click OK
8. Add Firewall exception by double clicking the IEDriverServer.exe file in its path "i.e., C:\Users\REX A JONES\Downloads\IEDriverServer"
9. Click Allow Access

Firefox

Firefox does not require a separate driver server like Google Chrome and Internet Explorer if using Selenium 2.0. There is an in-built Firefox server supported by Selenium 2.0. However, Firefox requires a separate driver called GeckoDriver to run Test Scripts in Selenium 3.x.

WebDriver uses GeckoDriver to control testing performed on Firefox. An error occurs in Eclipse if there is not a driver download. The driver can be downloaded directly from GitHub. In addition, users are directed to GitHub from Selenium HQ after clicking a link called Mozilla GeckoDriver. Follow the same steps (6 - 19) from Google Chrome's section after downloading GeckoDriver:

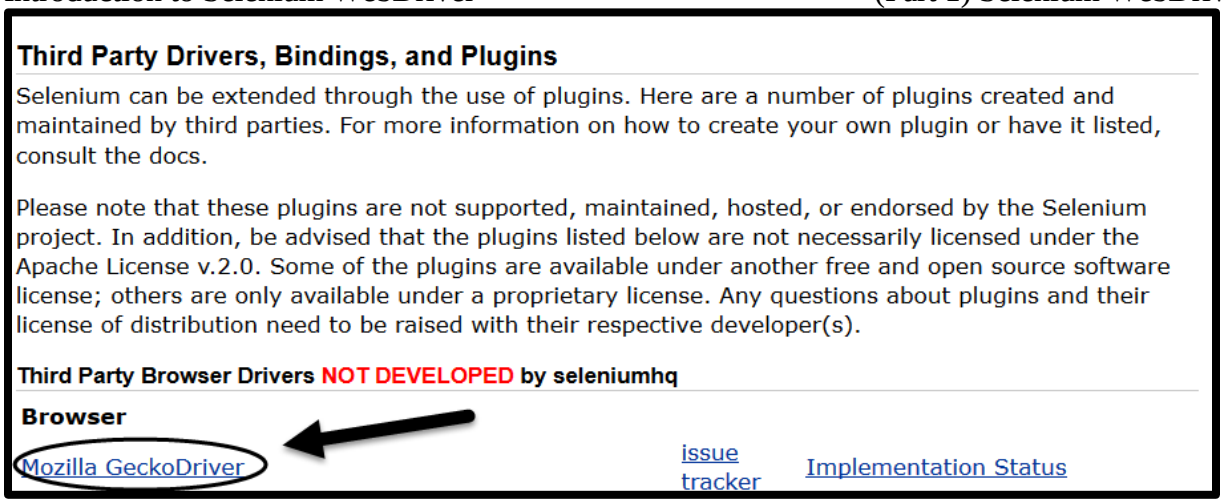


Figure 1.8 – Mozilla GeckoDriver

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

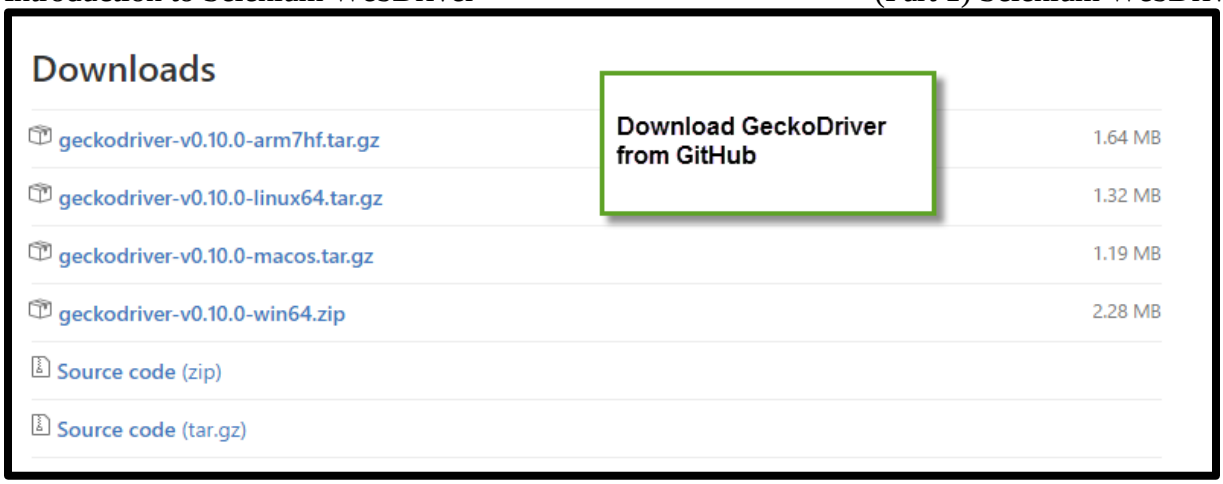


Figure 1.9 – Download GeckoDriver From GitHub

Download Notepad ++

Notepad ++ is a free source editor that supports multiple languages. Java uses Notepad ++ and other source editors to view files that end with a .java extension. The following are steps to download Notepad ++:

Steps To Download Notepad ++:

1. Go to https://notepad-plus-plus.org/download/v6.9.2.html

2. Click the Download button

3. Go to Download folder
4. Right click npp.6.9.2.Installer.exe then Select Open

Chapter 1 provided an overview of the upcoming chapters. The subsequent chapters explore WebDriver, WebElements, Selenium WebDriver Locator Types, and Selenium WebDriver Method Categories. Chapter 2 will explain WebDriver, WebElements, how to locate WebElements via HTML followed by finding and performing actions on WebElements.

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Chapter 2
WebDriver and WebElements

The starting point for testing all web applications begins with a browser. Selenium WebDriver provides support for major browsers such as Firefox, Google Chrome, and Internet Explorer. Calls are directed to the browser by using the browser's native support for automation. Common calls such as performing actions on WebElements are executed through a WebDriver.

A WebDriver is a collection of related methods. WebElements are buttons, text boxes, checkboxes, drop-down menus, hyperlinks, etc. on a web application. The Building Blocks for Selenium WebDriver is to first locate a WebElement then perform an action on the WebElement.

Chapter 2 will discuss the following regarding WebDriver and WebElements:

- ✓ WebDriver Packages and Classes
- ✓ WebDriver Object and Methods
- ✓ Locate WebElements via HTML
- ✓ Find and Perform Actions On WebElements

Note: A configuration for WebDriver is required before importing its packages. The driver object and methods are put to use after importing a WebDriver package. Chapter 1 – Configure WebDriver provides steps for configuring WebDriver.

WebDriver Packages and Classes

The WebDriver interface contains several classes. Each class is utilized through an imported package. In order to use WebDriver, the appropriate package must be imported to execute specific

commands. As an example, package “org.openqa.selenium.chrome” must be imported to use class “ChromeDriver” to execute a command that loads Google Chrome browser. The following is an example of importing a package for a Firefox driver:

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

WebDriver Object and Methods

Executions of the WebDriver commands are carried out through the driver object. The driver object is the main WebDriver component, which assist with locating and performing actions on WebElements. Like objects in Java programming, the driver object must be instantiated “created” before it is used. The following is an example of instantiating a Firefox driver object:

```
WebDriver driver = new FirefoxDriver();
```

After creating a driver object, the WebDriver methods are available to perform actions on a browser. An automation engineer can type the word “driver” followed by the dot operator (.) to view the methods. The following is a screenshot of the WebDriver methods:

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

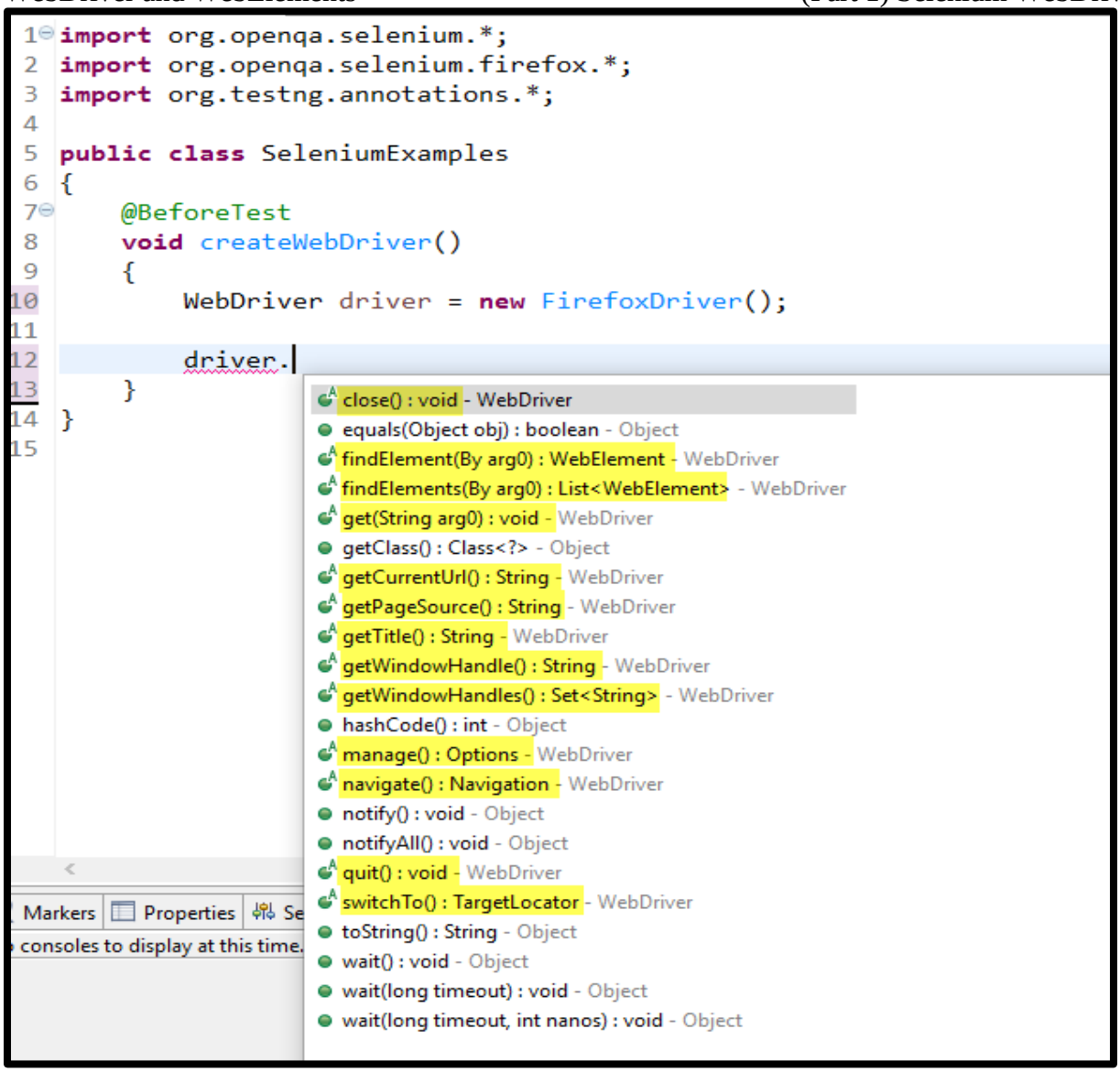


Figure 2.1 – WebDriver Methods

The WebDriver methods either return a value or return no value. For example, the findElement() method "**findElement(By arg0) : WebElement**" has a return value of "WebElement" which indicates a WebElement will be returned to the method. On the other hand, the close() method " **close() : void**" has a return value of "void" which indicates a value will not be returned. The same with get() method "**get(String arg0) : void**" which returns a value of void. However, the get() method receives a String parameter "**String arg0**". All strings have double quotes surrounding it's value. As mentioned in Chapter 1 – Selenium WebDriver Method Categories section, get() is a method that loads a new web page. The following is a complete syntax utilizing get() method for loading a new web page.

Note: driver.manage().window().maximize() is a statement for maximizing / enlarging the window.

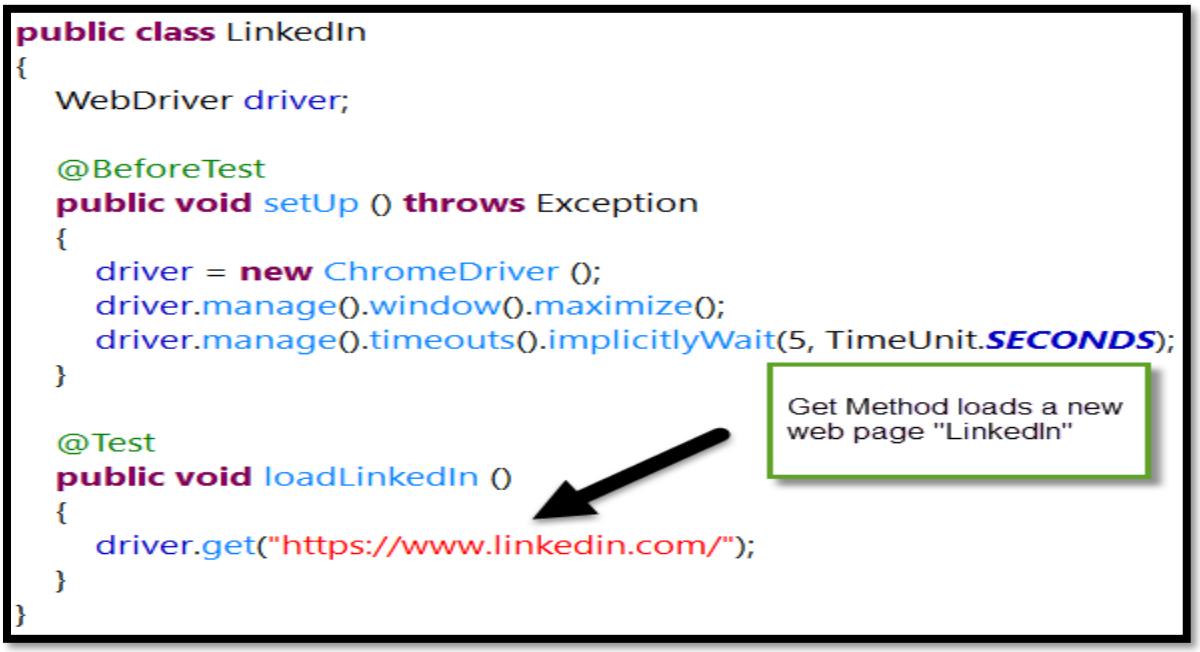


Figure 2.2 – Get Method Loads A New Web Page

The following describes each WebDriver Method:

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

WebDriver Method	Description
close()	Closes the current active window if there are multiple windows. The browser quits if only one window is active
findElement()	Find the first WebElement based on the locator type
findElements()	Find all elements within the current page based on the locator type
get()	Loads a new web page in the current browser window
getCurrentUrl()	Gets a string defining the current web page URL
getPageSource()	Gets the complete page source of the loaded web page.
getTitle()	Gets the current page title
getWindowHandle()	Handles a browser after switching a specific window
getWindowHandles()	Handles all browser windows and permits the user to switch control between the parent window and child window
manage()	Receives the option interface
navigate()	Navigates to a specific URL
quit()	Stops/Quits the driver instance and close all open browser windows
switchTo()	Switch from one browser window to another browser window

Figure 2.3 – WebDriver Methods and Descriptions

Locate WebElements via HTML

All web applications and browsers contain WebElements “i.e., buttons, links, etc.” The WebElements are derived from HyperText Markup Language (HTML). HTML is the standard markup language for creating web applications. Furthermore, each browser reads an HTML document to display the browser pages.

A great deal of HTML markup tags includes an opening (also known as starting) tag, a closing (also known as ending) tag, and at least one attribute. In regards to web applications and browsers, the markup tags determine the characteristics of a WebElement. Attributes provide additional infomation about the WebElement. It is important to know that attributes are defined in the opening tag.

Both tags have the same name surrounded by angle brackets (< >). However, the difference between an opening and closing tag is the forward slash (/). Closing tags must include a forward slash before the tag name. Unlike the tags, an attribute contains a name/value pair. The following is the syntax for an opening tag, closing tag, and attribute name/value pair.

Syntax
<Opening AttributeName=”AttributeValue”>
Content placed in between the tags
</Closing>

Note: Some elements do not contain a closing tag.

Find And Perform Actions On WebElements

WebDriver methods findElement() and findElements() are used to find WebElements. The difference between both methods are returned number of WebElements. WebDriver method findElement() returns one element while findElements() return multiple elements. If no elements are found then findElement() throws an exception “NoSuchElementException” but findElements() returns an empty list.

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

It is important to know a driver object must be created before searching for a WebElement. After creating a driver object, the driver object connects to one of the WebDriver methods via dot operator. The WebDriver method is followed by an open parenthesis, a By object, dot operator, and a locator type. The following is the syntax for findElement() and findElements():

Syntax for findElement()

```
driver.findElement(By.locatorType("value"));

// or

WebElement element = driver.findElement(By.id("value"));
```

Syntax for findElements()

```
driver.findElements(By.locatorType("value"));

// or

List <WebElement> elements = element.findElements(By.id("value"));
```

Note: Package "java.util.List" must be imported to write the last syntax which starts with "List <WebElement> …".

The object "By" is used to help locate WebElements. However, the object must be used in conjunction with a locator type. A locator type is the main component for finding and matching WebElelments. There are 8 locator types to find WebElements within an application. The following is a screenshot of all locator types in Eclipse after writing

```
driver.findElement(By.

// or

driver.findElements(By.
```

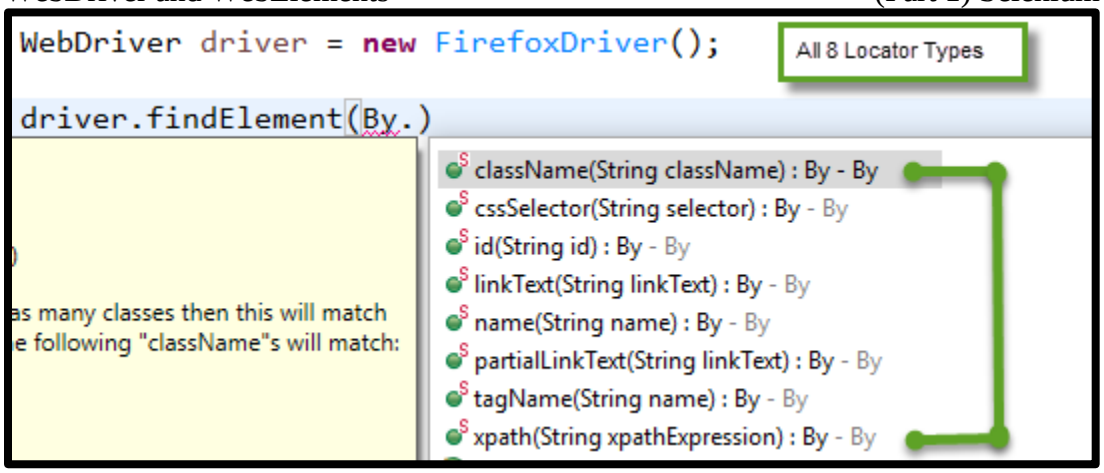


Figure 2.4 – Screenshot of All Locator Types

The following is a description of each locator type in alphabetical order:

Locator Type	Description
*By.**className***	Find WebElements by the value of its Class attribute
*By.**cssSelector***	Find WebElements by the CSS Selector's engine
*By.**id***	Find WebElements by the value of its ID attribute
*By.**linkText***	Find hyperlink WebElements by its complete text
*By.**name***	Find WebElements by the value of its Name attribute
*By.**partialLinkText***	Find hyperlink WebElements by partial text contained within the complete text
*By.**tagName***	Find WebElements by its tag name
*By.**xpath***	Find WebElements by its XPath

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Figure 2.5 – Eight Locator Types

An element can be located using a locator type if the HTML property (e.g., ID, Name, etc.) is present. Each locator type provides interaction between Selenium WebDriver and the web application.

After finding a WebElement, an action such as entering a value is performed on the WebElement. The following is a screenshot of available actions "WebElement Methods" in Eclipse after writing

```
driver.findElement(By.id("value")).

// or

WebElement element = driver.findElement(By.id("value"));
element.
```

Note: Both statements end with a dot operator (.).

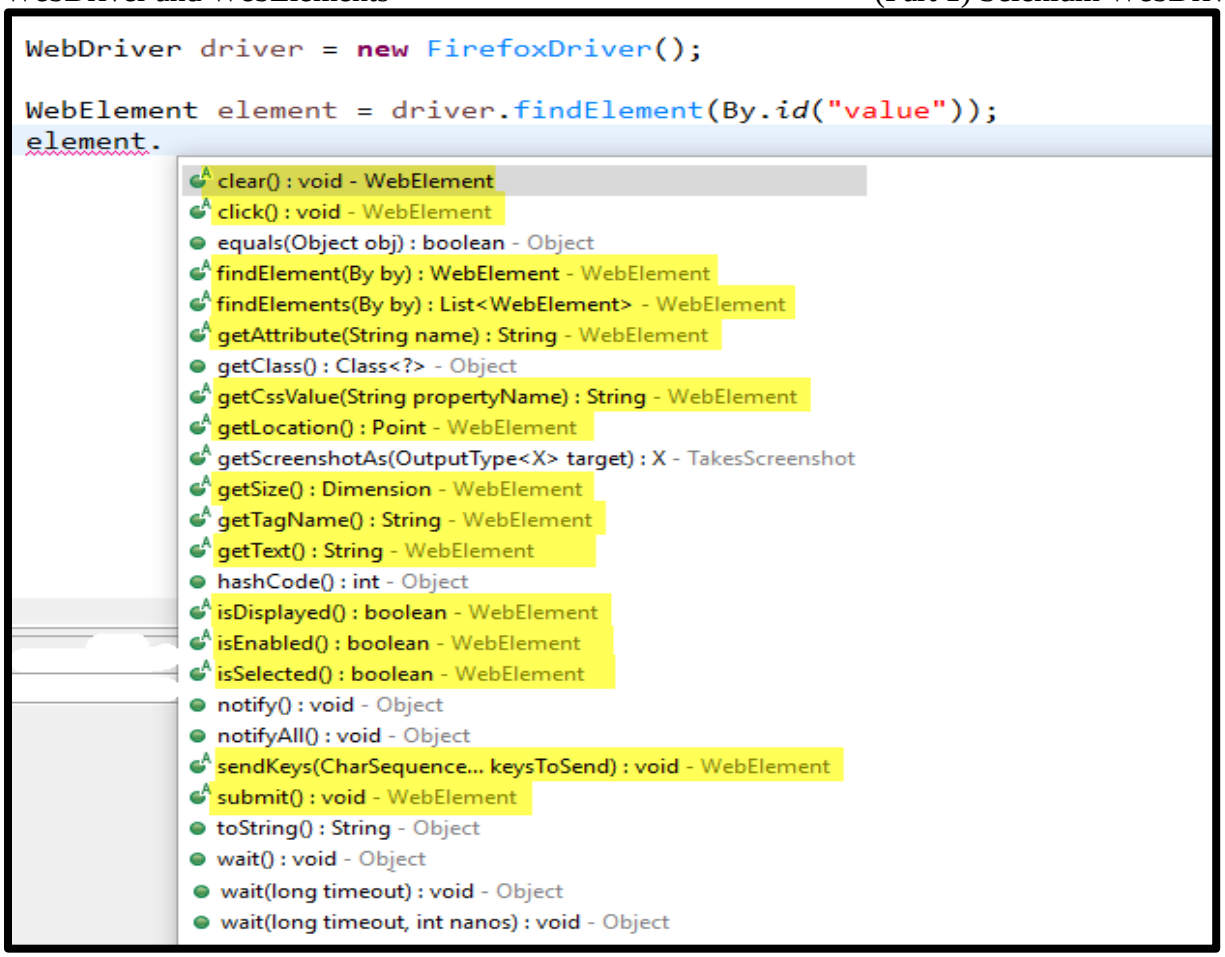


Figure 2.6 – Available Actions After Finding A WebElement

Chapter 2 discussed WebDriver and WebElements. WebDriver is an effort used for testing web applications. All web applications contain WebElements. WebElements are buttons, text boxes, checkboxes, drop-down menus, hyperlinks, etc. The key to testing a web application is finding the WebElement then performing an action on the WebElement. Chapters 3 through 10 explain how to find a WebElement by each locator type and provide examples for performing an action on the WebElement:

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Chapter 3
Find WebElement By ID

Finding an element by ID is one of the first priorities of all locator types. According to W3C, id attributes are unique on a web page. Therefore, the id attribute is extremely safe when locating an element within HTML. It is possible the developer will not change the ID when modifying a web page. The following is a screenshot of LinkedIn and its HTML markup tags for a User's First Name:

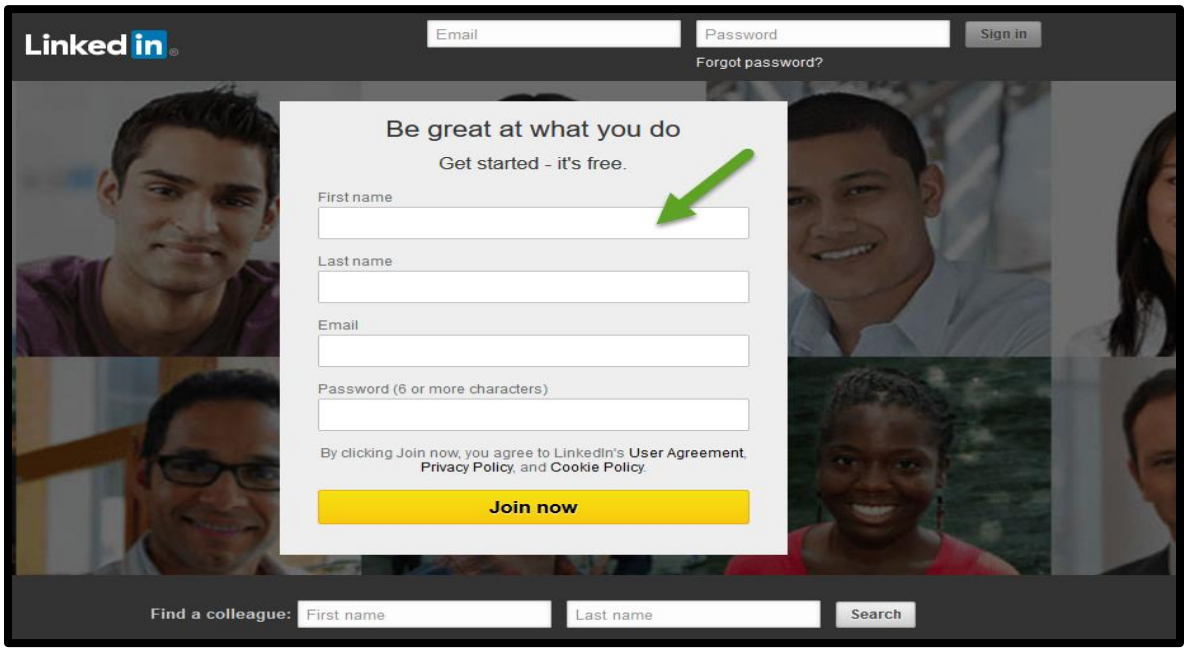


Figure 3.1 – LinkedIn's Home Screen (User's First Name)

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

```
<input id="first-name" class="cell-body-textinput" type="text" autofocus=""autofocus"/" value="" data-li-
bingo-validation="control" aria-required="true" name="firstName">
```

Figure 3.2 – HTML For LinkedIn's User First Name

The following is an example of finding LinkedIn's User First Name using the HTML id attribute:

```
driver.findElement(By.id("first-name"));

// or

WebElement userFirstName = driver.findElement(By.id("first-name"));
```

Enter Text In The User's First Name Text Field

The following is code for entering text in LinkedIn's User First Name text field via id attribute:

```
1  package LinkedInHomePage;
2  import org.openqa.selenium.*;
5
6  public class JoinLinkedIn1
7  {
8      WebDriver driver;
9
10     @BeforeTest
11     public void setUp() throws Exception
12     {
13         driver = new FirefoxDriver();
14
15         // Go to LinkedIn's Home Page
16         driver.get("https://www.linkedin.com/");
17     }
18
19     @AfterTest
20     public void tearDown() throws Exception
21     {
22         driver.quit();
23     }
24
25     @Test
26     void joinLinkedIn_1()
27     {
28         // Find first name via ID locator type
29         // Enter first name via send keys
30         driver.findElement(By.id("first-name")).sendKeys("Test First Name");
31     }
32 }
```

1. The **get()** method loads LinkedIn's Home page
2. The First Name WebElement is found using **findElement()** and locator type **By.*id***
3. Text is entered using **sendKeys()**
4. The **quit()** method closes the browser

Figure 3.3 – Perform Actions On The User's First Name Text Field (1)

- Line 8 "WebDriver driver" is the interface for driving the browser. Currently, the object reference variable "driver" points to nothing but will point to a FirefoxDriver object in a subsequent line "driver = **new** FirefoxDriver()".

- Line 13 "driver = **new** FirefoxDriver()" is an implementation of the WebDriver interface. The object reference variable "driver" is pointing to new FirefoxDriver() which means testing is controlled on the Firefox browser.

- Line 16 "driver.get("https://www.linkedin.com/")" loads a new LinkedIn Home page in the current browser window

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

- Line 22 "driver.quit()" quits the driver instance and closes the open browser window

- Line 30 "driver.findElement(By.*id*("first-name")).sendKeys("Test First Name")":
  - driver – WebDriver object reference variable that assist with finding a WebElement
  - findElement – a WebDriver method that finds the text field WebElement "First name" on LinkedIn's Home page
  - (By.*id*("first-name")) – By and *id* are parameters of the findElement WebDriver method. By is an object which locate elements and id is the locator type. The id locator type accepts a string parameter "first-name" which is the value of the HTML id attribute
  - sendKeys("Test First Name") – types the text "Test First Name" in the First Name text field

Note: An exception is an error that occurs at runtime. Lines 11 and 20 define an exception via **throws** Exception. It is used to explicitly identify an exception that the setUp() and tearDown() methods might come across at runtime. See Errors, Exceptions, and Debugging in Chapter 7 of "(Part 2) Java 4 Selenium WebDriver".

The following is a screenshot of LinkedIn's Home page after entering text in the User's First Name text field:

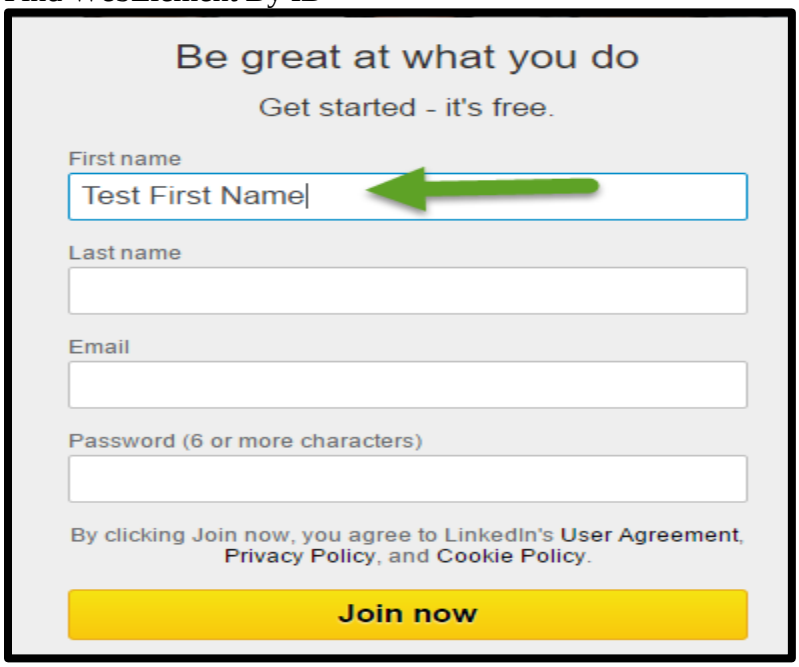


Figure 3.4 – Enter Text In The User's First Name Text Field

Enter and Clear Text In The User's First Name Text Field

The following is code for entering and clearing text in LinkedIn's User First Name text field via id attribute:

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

```
1  package LinkedInHomePage;
2  import org.openqa.selenium.*;
5
6  public class JoinLinkedIn2
7  {
8      WebDriver driver;
9
10     @BeforeTest
11     public void setUp() throws Exception
12     {
13         driver = new ChromeDriver();
14
15         // Go to LinkedIn's Home Page
16         driver.get("https://www.linkedin.com/");
17     }
18
19     @AfterTest
20     public void tearDown() throws Exception
21     {
22         driver.quit();
23     }
24
25     @Test
26     void enter_clearFirstName()
27     {
28         // Find first name via ID locator type
29         WebElement userFirstName = driver.findElement(By.id("first-name"));
30
31         // Enter first name via send keys
32         userFirstName.sendKeys("Test First Name");
33
34        // Clear first name via clear
35        userFirstName.clear();
36     }
37 }
```

1. The **get()** method loads LinkedIn's Home page

2. A **WebElement** object "userFirstName" is created then the first name WebElement is found using **findElement()** and locator type **By.*id***

3. Text is entered using **sendKeys()**

4. Text is cleared using **clear()**

5. The **quit()** method closes the browser

Figure 3.5 – Perform Actions On The User's First Name Text Field (2)

- Line 8 "WebDriver driver" is the interface for driving the browser. Currently, the object reference variable "driver" points to nothing but will point to a Chrome Driver object in a subsequent line "driver = new ChromeDriver()".

  Note: The System.*setProperty*() method was not used in this example since the ChromeDriver path was placed in Environment Variables – System Variables. However, the subsequent examples will contain an extra code line for System.*setProperty*() method:

Figure 3.6 – System.set*Property* () Prevents An Exception For Google Chrome

- The following are parameters for System.*setProperty*:
  - **key** = webdriver.chrome.driver
  - **value** = C:\\Users\\REX A JONES\\Downloads\\chromedriver_win32\\chromedriver.exe

- Line 13 "driver = **new** ChromeDriver()" is an implementation of the WebDriver interface. The object reference variable "driver" is pointing to new ChromeDriver() which means testing is controlled on the Chrome browser.

- Line 16 "driver.get("https://www.linkedin.com/")" loads a new LinkedIn Home page in the current browser window

- Line 22 "driver.quit()" quits the driver instance and closes the open browser window

- Line 29 "WebElement userFirstName = driver.findElement(By.*id*("first-name")):
  - WebElement userFirstName – creates an object reference variable called userFirstName that refers or points to a WebElement
  - driver – WebDriver object reference variable that assist with finding the WebElement
  - findElement – a WebDriver method that finds the text field WebElement "First name" on LinkedIn's Home page
  - (By.*id*("first-name")) – By and *id* are parameters of the findElement WebDriver method. By is an object which locate elements and id is the locator type. The id locator type accepts a string parameter "first-name" which is the value of the HTML id attribute

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

- Line 32 “userFirstName.sendKeys("Test First Name")” types the text “Test First Name” in the First Name text field via object reference variable “userFirstName”

- Line 35 “userFirstName.clear()” clears the text “Test First Name” from the First Name text field via object reference variable “userFirstName”

Note: This example was an illustration for clear() method. However, it is useful to create a WebElement object like userFirstName if multiple actions (i.e., enter text and clear text) are performed on the same WebElement (i.e., First Name text field).

The following is a screenshot of LinkedIn's Home page after clearing the User’s First Name:

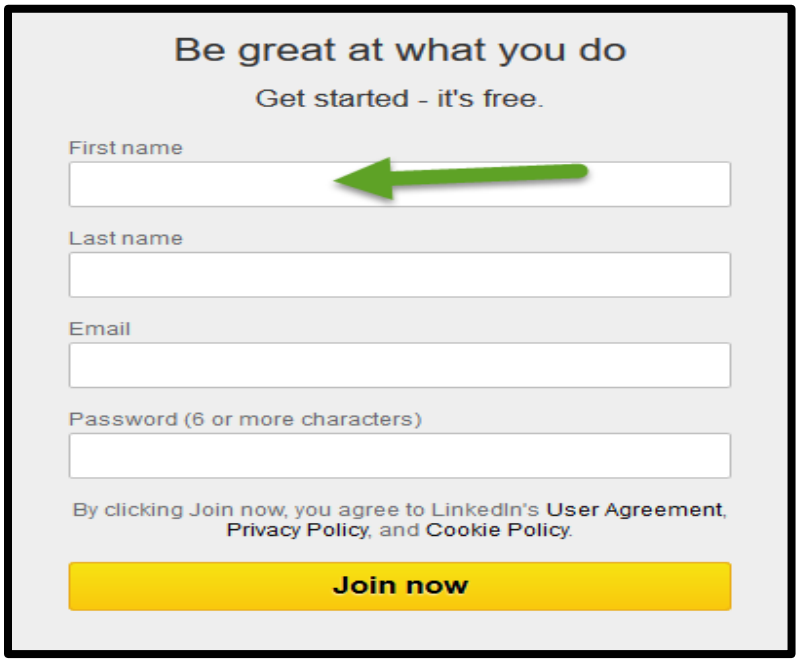


Figure 3.7 – Enter And Clear Text In The User’s First Name Text Field

Chapter 4
Find WebElement By Name

There are times when a web page may not contain an id attribute. If so, then an automation engineer can search for the value of a name attribute. Finding an element by name is similar to finding an element by the ID locator type. Usually, the name attribute is located in WebElements such as text fields and buttons. However, unlike the id attribute, which contains a unique value, the name attribute can contain the same value within HTML's source code. The following is a screenshot of LinkedIn and its HTML markup tags for a Colleague's First and Last Name:

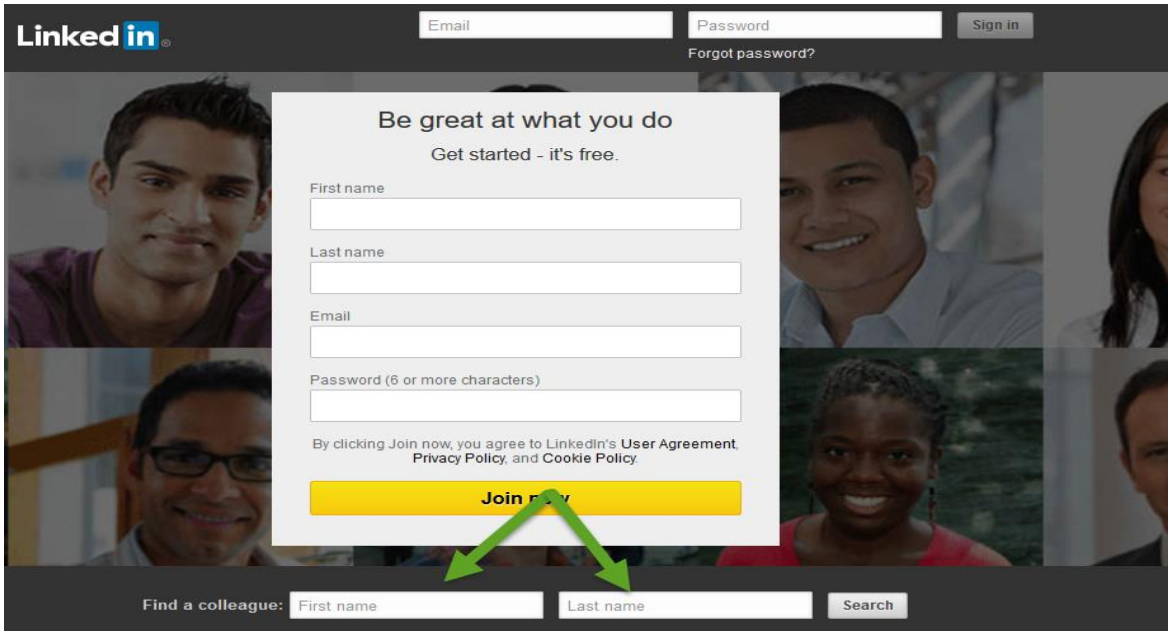


Figure 4.1 – LinkedIn's Home Screen (Colleague's First and Last Name)

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

```
<h3>Find a colleague:</h3>
<input type="text" placeholder="First name" name="first">
<input type="text" placeholder="Last name" name="last">
<input type="submit" value="Search" name="search">
```

Figure 4.2 – HTML For LinkedIn's Colleague First and Last Name

Note: The HTML markup tags do not contain an id attribute for Colleague's First or Last Name:

Enter Text In The Colleague's First and Last Name Text Fields

The following is code for entering text in LinkedIn's Colleague First and Last Name text fields via name attribute:

```java
package LinkedInHomePage;
import org.openqa.selenium.*;

public class FindColleague
{
    WebDriver driver;

    @BeforeTest
    public void setUp() throws Exception
    {
        System.setProperty("webdriver.chrome.driver", "C:\\Users\\REX A JONES\\Downloads\\chromedriver_win32\\chromedriver.exe");
        driver = new ChromeDriver();

        driver.get("https://www.linkedin.com/");
    }

    @AfterTest
    public void tearDown() throws Exception
    {
        driver.quit();
    }

    @Test
    public void enterColleagueName()
    {
        driver.findElement(By.name("first")).sendKeys("Rex");
        driver.findElement(By.name("last")).sendKeys("Jones");
    }
}
```

1. The **get()** method loads LinkedIn's Home page
2. The First Name WebElement is found using **findElement()** and locator type **By.*name***
3. The Last Name WebElement is found using **findElement()** and locator type **By.*name***
4. Text is entered using **sendKeys()** for First and Last Name
5. The **quit()** method closes the

Figure 4.3 – Perform Actions On The Colleague's First Name Text Field

- Line 8 "WebDriver driver" is the interface for driving the browser. Currently, the object reference variable "driver" points to nothing but will point to a Chrome Driver object in a subsequent line "driver = new ChromeDriver()".

- Line 13 tells Selenium where the executable file for Chrome driver is located via System.*setProperty*. The executable file operates like a bridge between Selenium and the browser. All browsers except Firefox require an executable file. Steps for downloading the executable file are located in Chapter 1 – Download Browser Drivers section. The following are parameters for System.*setProperty*:
  - **key** = webdriver.chrome.driver
  - **value** = C:\\Users\\REX A JONES\\Downloads\\chromedriver_win32\\chromedriver.exe

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Note**:** Value is the path location of the executable file

- Line 14 "driver = **new** ChromeDriver()" is an implementation of the WebDriver interface. The object reference variable "driver" is pointing to new ChromeDriver() which means testing is controlled on the Chrome browser.

- Line 16 "driver.get("https://www.linkedin.com/")" loads a new LinkedIn Home page in the current browser window

- Line 22 "driver.quit()" quits the driver instance and closes the open browser window

- Line 28 "driver.findElement(By.*name*("first")).sendKeys("Rex")":
  - driver – WebDriver object reference variable that assist with finding a WebElement
  - findElement – a WebDriver method that finds the text field WebElement "First name" on LinkedIn's Home page
  - (By.*name*("first")) – By and *name* are parameters of the findElement WebDriver method. By is an object which locate elements and name is the locator type. The name locator type accepts a string parameter "first" which is the value of the HTML name attribute
  - sendKeys("Rex") – types the text "Rex" in the Colleague's First Name text field

- Line 29 "driver.findElement(By.*name*("last")).sendKeys("Jones")":
  - driver – WebDriver object reference variable that assist with finding a WebElement
  - findElement – a WebDriver method that finds the text field WebElement "First name" on LinkedIn's Home page
  - (By.*name*("last")) – By.name are parameters of the findElement WebDriver method. By is an object which locate elements and name is the locator type. The name locator type accepts a string parameter "last" that is the value of the HTML name attribute
  - sendKeys("Jones") – types the text "Jones" in the Colleague's Last Name text field

The following is a screenshot of LinkedIn's Home page after entering text in the Colleague's First and Last Name field:

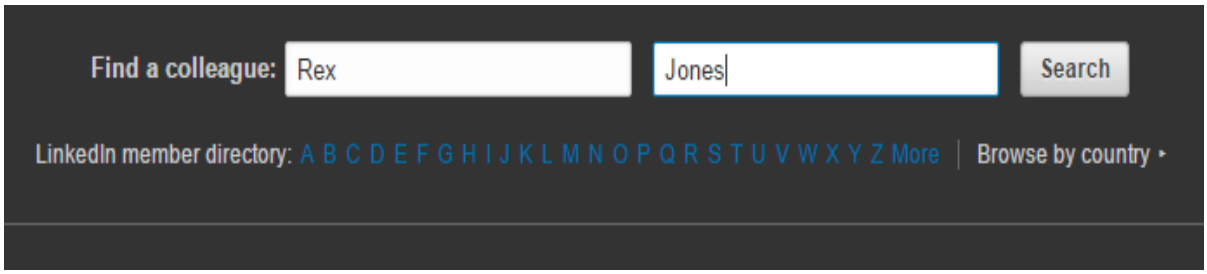


Figure 4.4 – Enter Text In The Colleague's First and Last Name Text Field

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Chapter 5
Find WebElement By XPath

XML Path Language “XPath” is a powerful method for finding WebElements. The XPath locator type refers to a specific node in the DOM tree. An automation engineer can manually create an XPath using the web page’s HTML or an inspector tool that generates the XPath. The XPath locator accepts two types of parameter strings to find a WebElement.

1. Absolute XPath – generates a complete XPath starting from the <html> tag or single forward slash (/)
2. Relative XPath – generates a shortened version of the XPath starting with double forward slashes (//)

Below are screenshots of Facebook's Sign Up page and its markup tags for radio buttons “Female and Male” followed by Absolute XPath values:

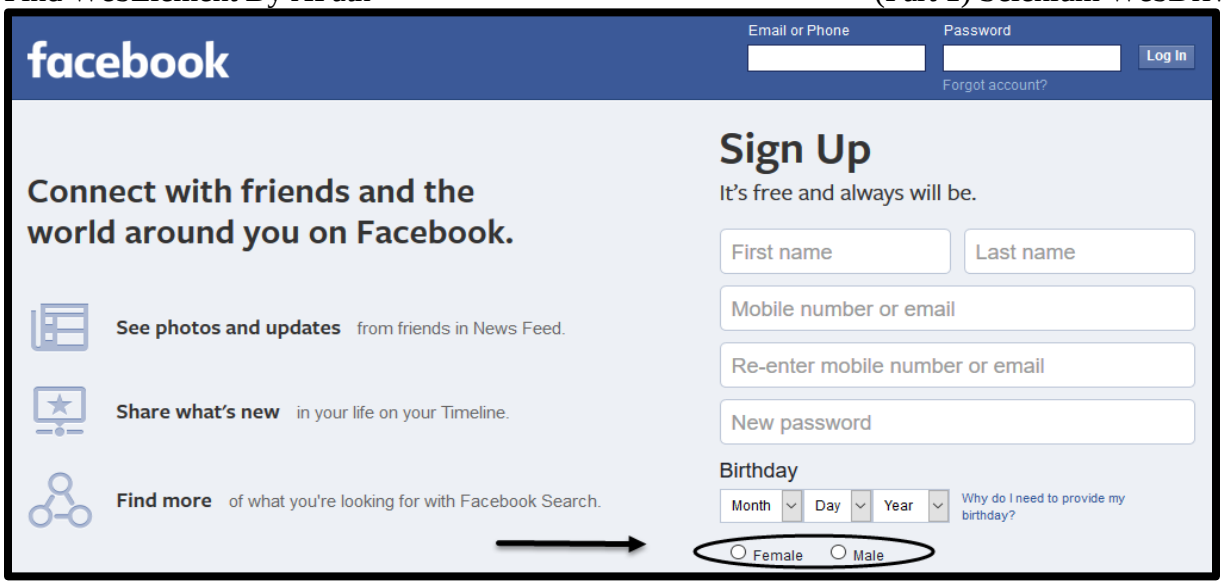


Figure 5.1 – Facebook's Sign Up Page / Female and Male Radio Buttons

```
<span id="u_0_h" class="_5k_3" data-name="gender_wrapper" data-type="radio">
  <span class="_5k_2 _5dba">
      <input id="u_0_e" type="radio" value="1" name="sex"/>
      <label class="_58mt" for="u_0_e">Female</label>
  </span>
  <span class="_5k_2 _5dba">
      <input id="u_0_f" type="radio" value="2" name="sex"/>
      <label class="_58mt" for="u_0_f">Male</label>
  </span>
</span>
```

Figure 5.2 – HTML For Female and Male Radio Buttons On Facebook's Sign Up Page (1)

Result: The XPath text box displays a string parameter for absolute XPath.

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

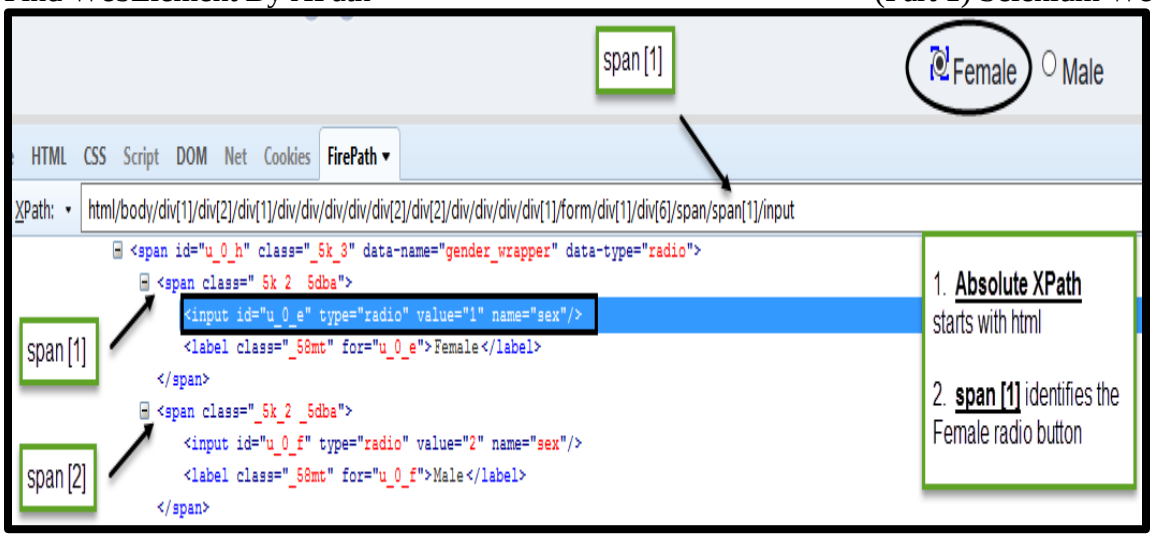


Figure 5.3 – Absolute XPath For The Female Radio Button

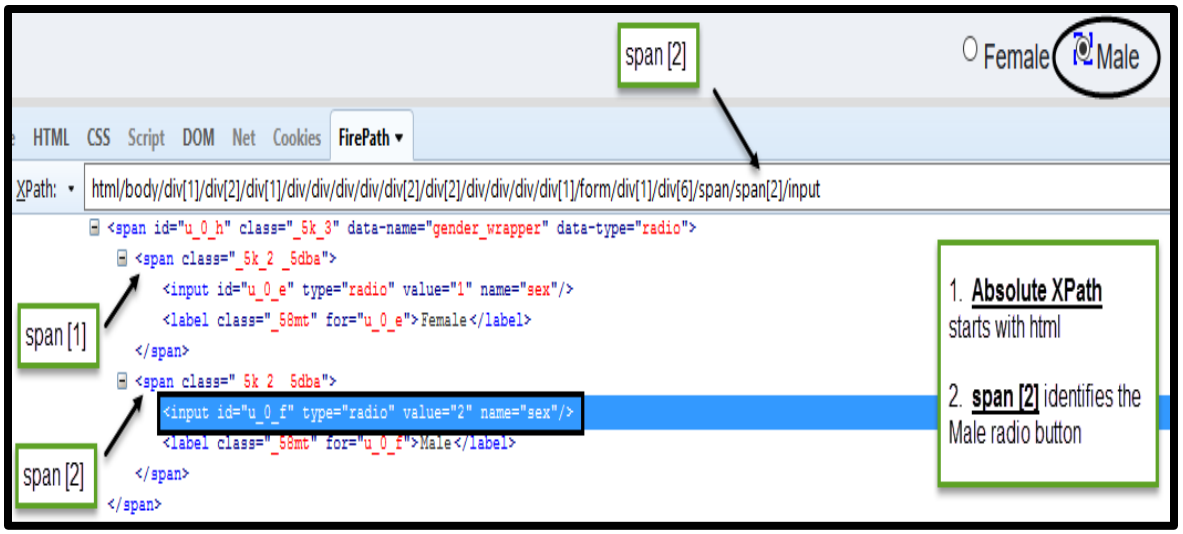


Figure 5.4 – Absolute XPath For The Male Radio Button

Figure 5.5 – Relative XPath For The Female Radio Button

Figure 5.6 – Relative XPath For The Male Radio Button

The relative XPath finds the nearest id attribute for the chosen WebElement. In this case, the value for Female id attribute is u_0_e and Male id attribute is u_0_f. However, the id attribute value of "u_0_h" would have been chosen as the relative XPath if the radio buttons "Female & Male" did not contain an id attribute.

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Chapter 5
Find WebElement By XPath (Part 1) Selenium WebDriver

It is recommended to use the relative XPath rather than the absolute XPath. Both XPaths are valuable but the absolute XPath is more vulnerable to an HTML code change. The chosen WebElement "i.e., radio button" becomes invalid if a developer adds or removes a tag prior to the WebElement. For example, the Test Script will not be functional if a developer modifies the HTML code "i.e., div tag" prior to span[1] in Figure 5.3.

Select A Radio Button via Relative XPath

The following is code for selecting the Female radio button on Facebook's Sign Up Page via XPath locator type:

```
1  package FacebookSignUpPage;
2  import org.openqa.selenium.By;
6
7  public class SelectRadioButton
8  {
9      WebDriver driver;
10
11     @BeforeTest
12     public void setUp () throws Exception
13     {
14         System.setProperty("webdriver.chrome.driver", "C:\\Users\\REX A JONES\\Downloads\\chromedriver_win32\\chromedriver.exe");
15         driver = new ChromeDriver ();
16
17         driver.get("https://www.facebook.com/");
18     }
19
20     @AfterTest
21     public void tearDown () throws Exception
22     {
23         driver.quit();
24     }
25
26     @Test
27     public void selectGender ()
28     {
29         driver.findElement(By.xpath("//*[@id='u_0_e']")).click();
30     }
31 }
```

1. The **get()** method loads Facebook's Sign Up page
2. The Radio Button WebElement is found using **findElement()** and locator type **By.*xpath***
3. The radio button is clicked using **click()**
4. The **quit()** method closes the

Figure 5.7 – Perform Actions On The Radio Button

- Line 9 "WebDriver driver" is the interface for driving the browser. Currently, the object reference variable "driver" points to nothing but will point to a Chrome Driver object in a

subsequent line "driver = new ChromeDriver()".

- Line 14 tells Selenium where the executable file for Chrome driver is located via System.*setProperty*. The executable file operates like a bridge between Selenium and the browser. All browsers except Firefox require an executable file. Steps for downloading the executable file are located in Chapter 1 – Download Browser Drivers section. The following are parameters for System.*setProperty*:
  - **key** = webdriver.chrome.driver
  - **value** = C:\\Users\\REX A JONES\\Downloads\\chromedriver_win32\\chromedriver.exe
    Note**:** Value is the path location of the executable file

- Line 15 "driver = **new** ChromeDriver()" is an implementation of the WebDriver interface. The object reference variable "driver" is pointing to new ChromeDriver() which means testing is controlled on the Chrome browser.

- Line 17 "driver.get("https://www.facebook.com/")" loads a new Facebook page in the current browser window

- Line 23 "driver.quit()" quits the driver instance and closes the open browser window

- Line 29 "driver.findElement(By.*xpath*("//*[@id='u_0_e']")).click()":
  - driver – WebDriver object reference variable that assist with finding a WebElement
  - findElement – a WebDriver method that finds the radio button WebElement "Female" on Facebook's Sign Up page
  - (By.*xpath*("//*[@id='u_0_e']")) – By and *xpath* are parameters of the findElement WebDriver method. By is an object which locate elements and xpath is the locator type. The XPath locator type accepts a string parameter "//*[@id='u_0_e']" which is the relative XPath for the Female radio button.
  - click() – clicks the Female radio button
- Relative XPath contains the dot operator "**.//*[@id='u_0_e']**"
- Test Script removes the dot operator (By.*xpath*("//*[@id='u_0_e']"))

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

The following is a screenshot of Facebook's Sign Up page after selecting the Female radio button:

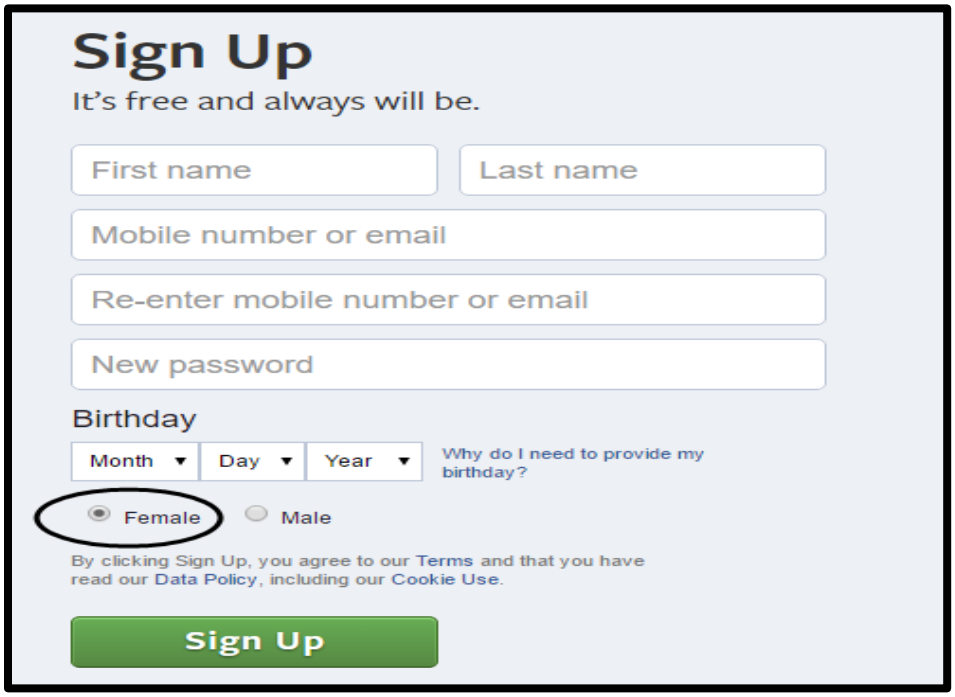


Figure 5.8 – Select The Female Radio Button

Relative XPath Alternatives

The following HTML screenshot of Facebook's Female and Male radio buttons will be used to explain alternatives for executing relative XPath:

```
<span id="u_0_h" class="_5k_3" data-name="gender_wrapper" data-type="radio">
  <span class="_5k_2 _5dba">
      <input id="u_0_e" type="radio" value="1" name="sex"/>
      <label class="_58mt" for="u_0_e">Female</label>
  </span>
  <span class="_5k_2 _5dba">
      <input id="u_0_f" type="radio" value="2" name="sex"/>
      <label class="_58mt" for="u_0_f">Male</label>
  </span>
</span>
```

Figure 5.9 – HTML For Female and Male Radio Buttons On Facebooks's Sign Up Page (2)

The relative xpath's locator (By.*xpath*("//*[@id='u_0_e']")) was used to click the Female radio button. An asterisk (*) in the relative XPath represents any tag within the HTML code. Therefore, an automation engineer can choose to be more specific and replace the asterisk with a tag name. The following is the syntax for finding a WebElement by relative XPath:

Syntax
//TagName[@AttributeName='AttributeValue']

The following code shows how input is a suitable tag name for clicking the radio button.

(By.*xpath*("//input[@id='u_0_e']"))

According to the syntax, the attribute "id" can be replaced with any attribute. An attribute such as name or type that identifies the WebElement is suitable. However, there are attributes/values that are duplicated within HTML's source code. If duplicate attributes/values exist for a WebElement then an automation engineer can use multiple attributes. Multiple attributes are beneficial if it is difficult find a unique attribute/value. The following are examples for utilizing a single attribute and multiple attributes:

- (By.*xpath*("//*[@for='u_0_e']"))
- (By.*xpath*("//label[@for='u_0_e']"))

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

- (By.*xpath*("//*[@type='radio' and @value='1']"))
- (By.*xpath*("//input[@type='radio' and @name='sex' and @value='1']"))

Working With Dynamic WebElements Using XPath

There are situations when a web application may contain dynamic WebElements. A dynamic WebElement is when the value on an application changes at runtime. For example, the time will change every second/minute on an application during runtime.

In addition, a dynamic WebElement manifests when the value of an attribute changes upon reloading the web application. The Home page of Yahoo is a perfect example of a dynamic WebElement. All of the list box WebElements change when the page reloads. A user has to search for information, select an option from the list of options, and then return to Yahoo's Home page. The following screenshots show how the same set of options contains a different value for ID attribute:

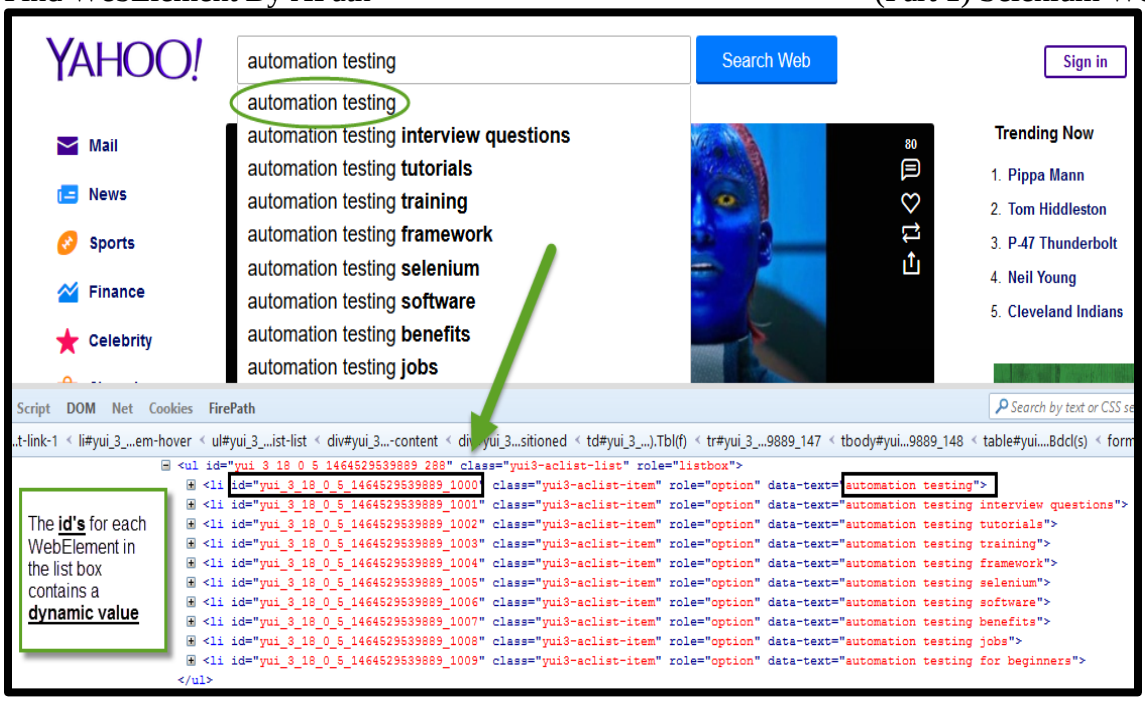


Figure 5.10 – HTML For Yahoo's Home Page Listbox / Initial Values

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

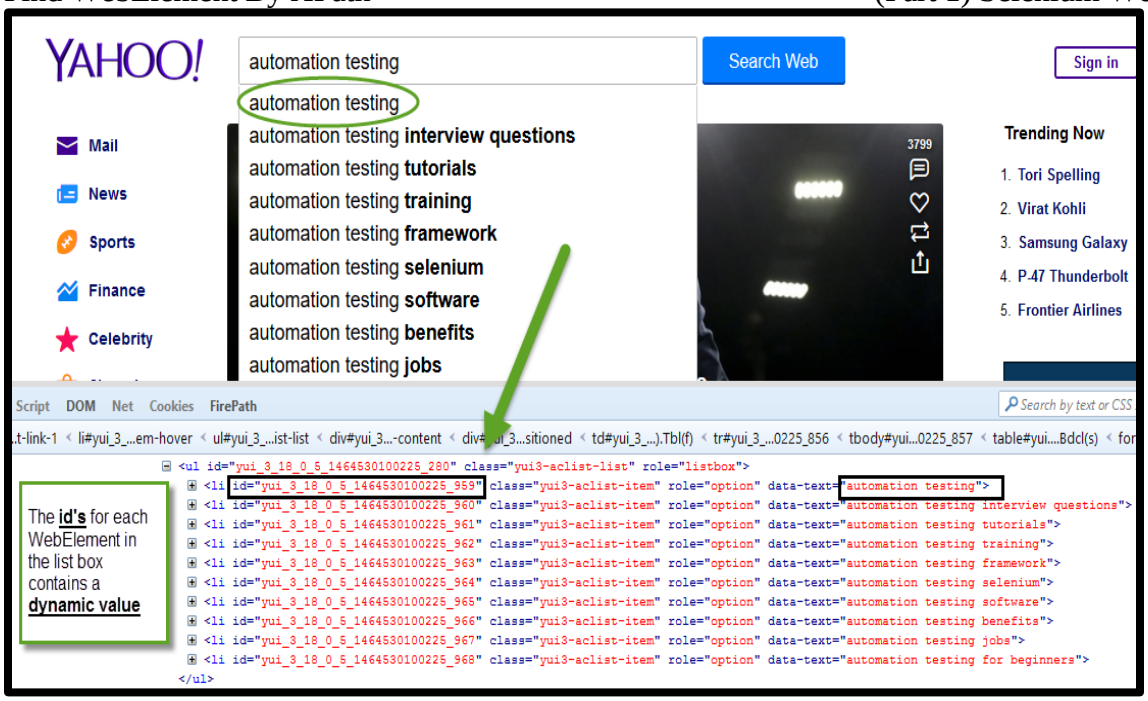


Figure 5.11 – HTML For Yahoo's Home Page Listbox / Updated Values

The screenshots display a dynamic value for ID attribute. In particular, the ID changes for list box option "automation testing" from "yui_3_18_0_5_1464529539889_1000" to "yui_3_18_0_5_1464530100225_959".

An exception occurs if the locator attempts to find a dynamic element. The Test Script will not find the element because the value changed. Nevertheless, the Relative XPath can find dynamic WebElements utilizing a partial pattern match. A partial pattern match locates a WebElement based on part of the value. The following is the syntax for using a partial pattern match via relative XPath:

Syntax
[PartialPatternMatch(@AttributeName,'AttributeValue')]

Chapter 5
Find WebElement By XPath (Part 1) Selenium WebDriver

Assume the value for an ID attribute changes from “Test123” to “Test456”, “123Test” to “456Test”, and “123Test456” to “789Test123”. The constant value in each example is Test. As a result, an automation engineer can manually write a partial pattern match to find the dynamic WebElement. The following are three partial pattern match types that assist with dynamic values: starts-with, ends-with, contains

1. **starts-with** – finds a WebElement if the value starts with a constant string value “Test123” and “Test456”

   driver.findElement.(By.*xpath*("//[starts-with(@id,'Test')]”))

2. **ends-with** – finds a WebElement if the value ends with a constant string value “123Test” and “456Test”

   driver.findElement.(By.*xpath*("//[ends-with(@id,'Test')]”))

3. **contains** – finds a WebElement if the value contains a constant string value “123Test456” and “789Test123”

   driver.findElement.(By.*xpath*("//[contains(@id,'Test')]”))

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Chapter 6
Find WebElement By CSS Selector

Cascade Style Sheet “CSS” Selector is a robust method for finding WebElements. This locator type is very similar to the XPath locator type. An automation engineer can manually create a CSS Selector from the web page’s HTML. Below are screenshots of Facebook's Sign Up page and its markup tags for radio buttons “Female and Male” followed by CSS Selectors.

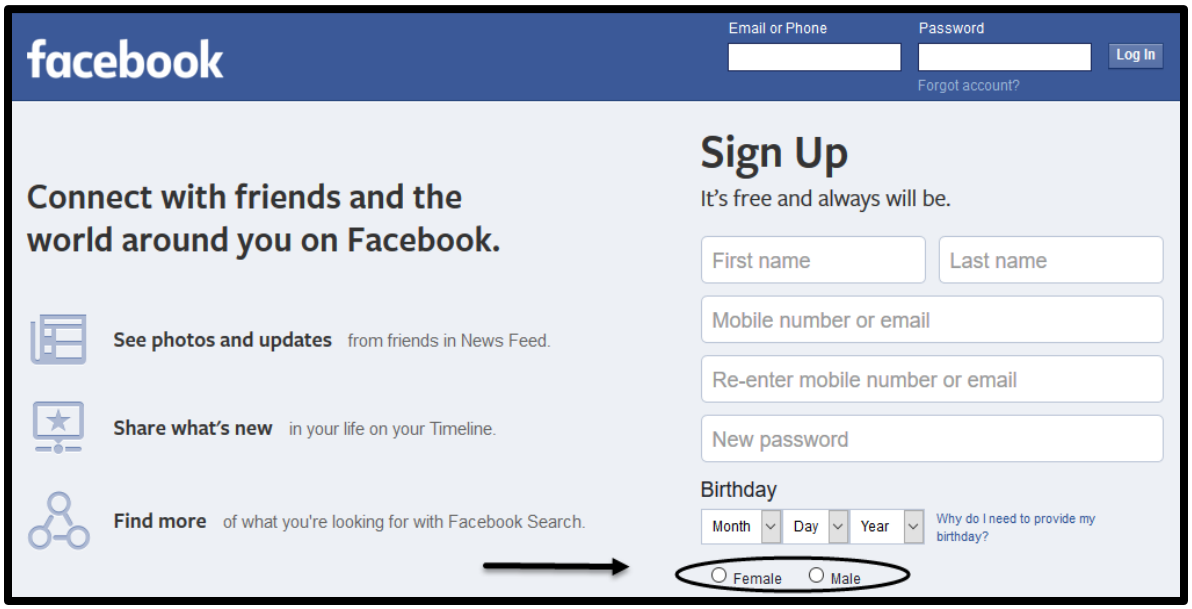


Figure 6.1 – Facebook’s Sign Up Page / Female and Male Radio Buttons

```
<span id="u_0_h" class="_5k_3" data-name="gender_wrapper" data-type="radio">
  <span class="_5k_2 _5dba">
      <input id="u_0_e" type="radio" value="1" name="sex"/>
      <label class="_58mt" for="u_0_e">Female</label>
  </span>
  <span class="_5k_2 _5dba">
      <input id="u_0_f" type="radio" value="2" name="sex"/>
      <label class="_58mt" for="u_0_f">Male</label>
  </span>
</span>
```

Figure 6.2 – HTML For Female and Male Radio Buttons On Facebook's Sign Up Page

Figure 6.3 – CSS Selector For The Female Radio Button

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Chapter 6
Find WebElement By CSS Selector (Part 1) Selenium WebDriver

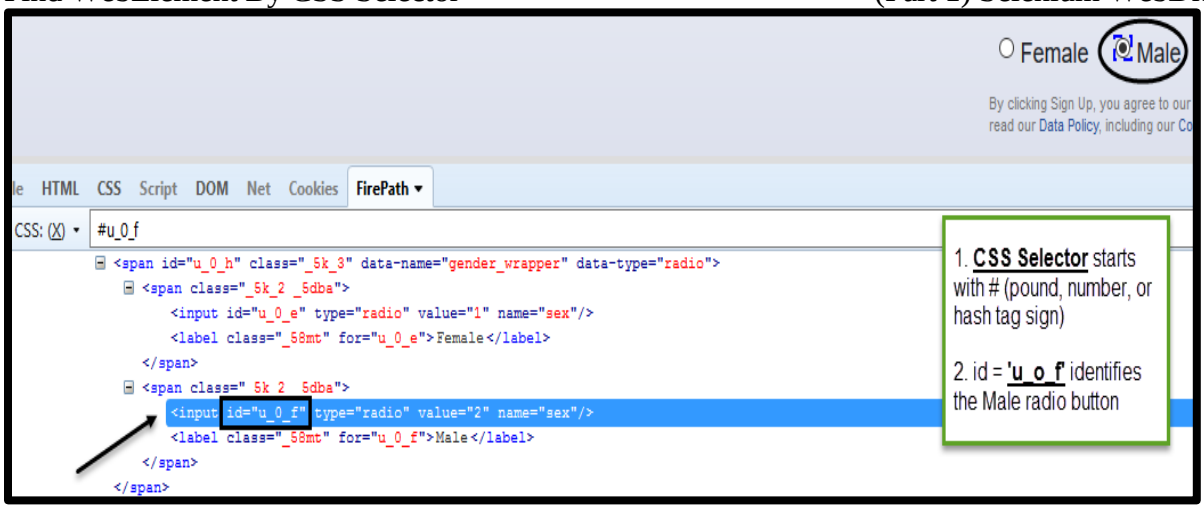


Figure 6.4 – CSS Selector For The Male Radio Button

Select A Radio Button via CSS Selector

The following is code for selecting the Male radio button on Facebook's Sign Up Page via CSS Selector locator type:

```
1  package FacebookSignUpPage;
2  import org.openqa.selenium.By;
6
7  public class SelectRadioButton2
8  {
9      WebDriver driver;
10
11     @BeforeTest
12     public void setUp () throws Exception
13     {
14         System.setProperty("webdriver.chrome.driver", "C:\\Users\\REX A JONES\\Downloads\\chromedriver_win32\\chromedriver.exe");
15         driver = new ChromeDriver ();
16
17         driver.get("https://www.facebook.com/");
18     }
19
20     @AfterTest
21     public void tearDown () throws Exception
22     {
23         driver.quit();
24     }
25
26     @Test
27     public void selectGender ()
28     {
29         driver.findElement(By.cssSelector("#u_0_f")).click();
30     }
31 }
```

1 4 2 3

1. The **get()** method loads Facebook's Sign Up page
2. The Radio Button WebElement is found using **findElement()** and locator type **By.*cssSelector***
3. The radio button is clicked using **click()**
4. The **quit()** method closes the

Figure 6.5 – Perform Actions On The Radio Button

- Line 9 "WebDriver driver" is the interface for driving the browser. Currently, the reference "driver" points to nothing but will point to a Chrome Driver object in a subsequent line "driver = new ChromeDriver()".

- Line 14 tells Selenium where the executable file for Chrome driver is located via System.*setProperty*. The executable file operates like a bridge between Selenium and the browser. All browsers except Firefox require an executable file. Steps for downloading the executable file are located in Chapter 1 – Download Browser Drivers section. The following are parameters for System.*setProperty*:
  - **key** = webdriver.chrome.driver
  - **value** = C:\\Users\\REX A JONES\\Downloads\\chromedriver_win32\\chromedriver.exe
    Note**:** Value is the path location of the executable file

- Line 15 "driver = **new** ChromeDriver()" is an implementation of the WebDriver interface. The reference "driver" is pointing to new ChromeDriver() which means testing is controlled

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

on the Chrome browser.

- Line 17 “driver.get("https://www.facebook.com/")” loads a new Facebook page in the current browser window

- Line 23 “driver.quit()” quits the driver instance and closes the open browser window

- Line 29 “driver.findElement(By.*cssSelector*("#u_0_f")).click();”:
  - driver – WebDriver object which assist with finding a WebElement
  - findElement – a WebDriver method that finds the WebElement “Male” radio button on Facebook’s Sign Up page
  - (By.*cssSelector*("#u_0_f")) – By and *cssSelector* are parameters of the findElement WebDriver method. By is an object which locate elements and *cssSelector* is the locator type. The CSS Selector locator type accepts a string parameter “#u_0_f” which is the CSS Selector for the Male radio button.
  - click() – clicks the Male radio button

The following is a screenshot of Facebook’s Sign Up page after selecting the Male radio button:

Figure 6.6 – Select The Male Radio Button

CSS Selector Alternatives

The number "#" (also known as pound or hashtag) symbol starts a CSS Selector representing an ID attribute. In the previous example, CSS Selector (By.*cssSelector*("#u_0_f")) means the ID attribute's value is "u_0_f" for Male radio button. However, the dot (.) operator is a symbol that represents the Class attribute. The following is the syntax for identifying all elements using class if hypothetically the class name is Automation:

Syntax
(By.*cssSelector*(".Automation"))

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Chapter 6
Find WebElement By CSS Selector (Part 1) Selenium WebDriver

The following is the syntax for identifying an element using class if the tag type is Test and class name is Automation:

Syntax

(By.*cssSelector*("Test.Automation"))

The first syntax only used a dot operator with the class name (Automation). Therefore, all elements are identified if the class name is Automation. The second syntax used the tag name (Test), dot operator (.), and class name (Automation). A specific element will be identified since the Test Script contains a tag name and class name. CSS Selector has a syntax similar to relative XPath that identify WebElements. The difference between the CSS Selector and relative XPath is the two forward slashes and the at symbol "@" before attribute name:

CSS Selector Syntax

TagName[AttributeName='AttributeValue']

Relative XPath

//TagName[@AttributeName='AttributeValue']

The following is a screenshot of WordPress and its HTML markup tags for Remember Me checkbox:

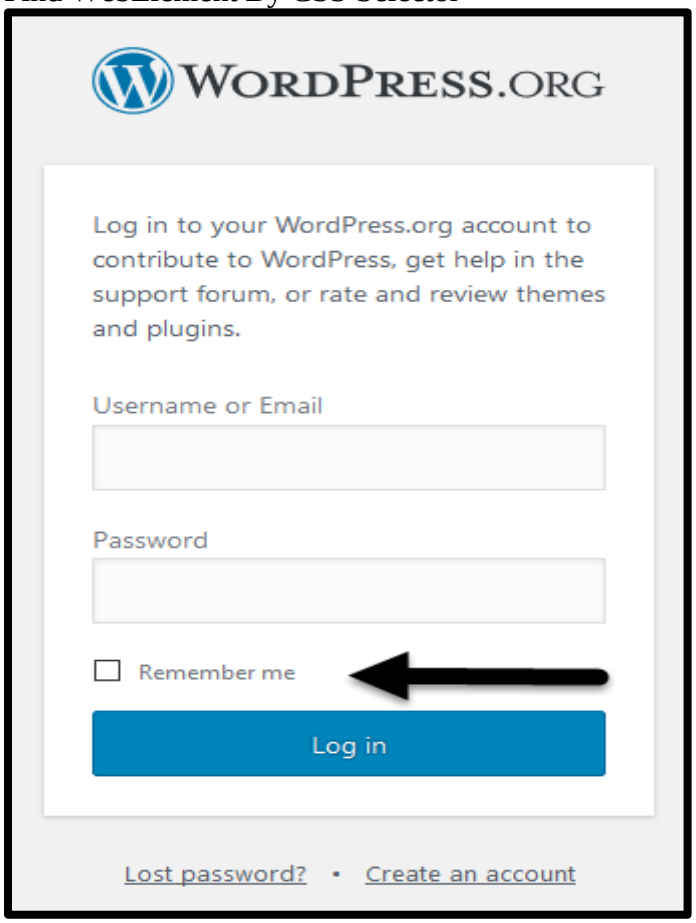


Figure 6.7 – WordPress Remember Me Checkbox

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

```
<p class="login-remember">
 <label>
      <input id="rememberme" type="checkbox" value="forever" name="rememberme">
     Remember me
  </label>
</p>
```

Figure 6.8 – HTML For The Remember Me Checkbox On WordPress Log In Page

The following table compares CSS Selector and relative XPath using the ID attribute for Remember Me checkbox:

CSS Selector	XPath
*[id='rememberme']	//*[@id='rememberme']
input[id='rememberme']	//input[@id='rememberme']

Figure 6.9 – Similarities Between CSS Selector and Relative XPath

Working With Dynamic WebElements Using CSS Selector

As mentioned earlier, a dynamic WebElement manifests when the value on an application changes or the value of an HTML attribute changes. The WebElement can be found although the value changes at runtime or after a page reloads. CSS Selector has three pattern matching symbols (^, $, *) that assist with finding dynamic WebElements. The following is the syntax for finding dynamic WebElements via CSS Selector:

Syntax

TagName[AttributeName Symbol='AttributeValue']

Chapter 6
Find WebElement By CSS Selector (Part 1) Selenium WebDriver

Assume the value for an ID attribute changes from “Hello123” to “Hello456”, “123Hello” to “456Hello”, and “123Hello456” to “789Hello123”. The constant value in each example is Hello. As a result, an automation engineer can insert a symbol (^, $, *) to find the dynamic WebElement. The following are three CSS Selector pattern matching symbols that find dynamic values:

1. ^ (caret) – finds a WebElement if the value starts with a constant string value “Hello123” and “Hello456”

   driver.findElement.(By.*cssSelector*("*[id ^=’Hello’]"))

2. **$** (dollar sign) – finds a WebElement if the value ends with a constant string value “123Hello” and “456Hello”

   driver.findElement.(By.*cssSelector*("*[id $=’Hello’]"))

3. ***** (asterisk) – finds a WebElement if the value contains a constant string value “123Hello456” and “789Hello123”

   driver.findElement.(By.*cssSelector*("*[id *=’Hello’]"))

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Chapter 7
Find WebElement By Link Text

Finding an element by Link Text is carried out via the hyperlink's text name. Using a hyperlink's text name is the easiest way to click a hyperlink. The entire hyperlink's text name is used as the string parameter for the Link Text locator type. It is important to know that the hyperlink's text name is placed within HTML's anchor 'a' tags and after the href attribute/value. The following is a screenshot of LinkedIn and its HTML markup tags for the Forgot Password hyperlink:

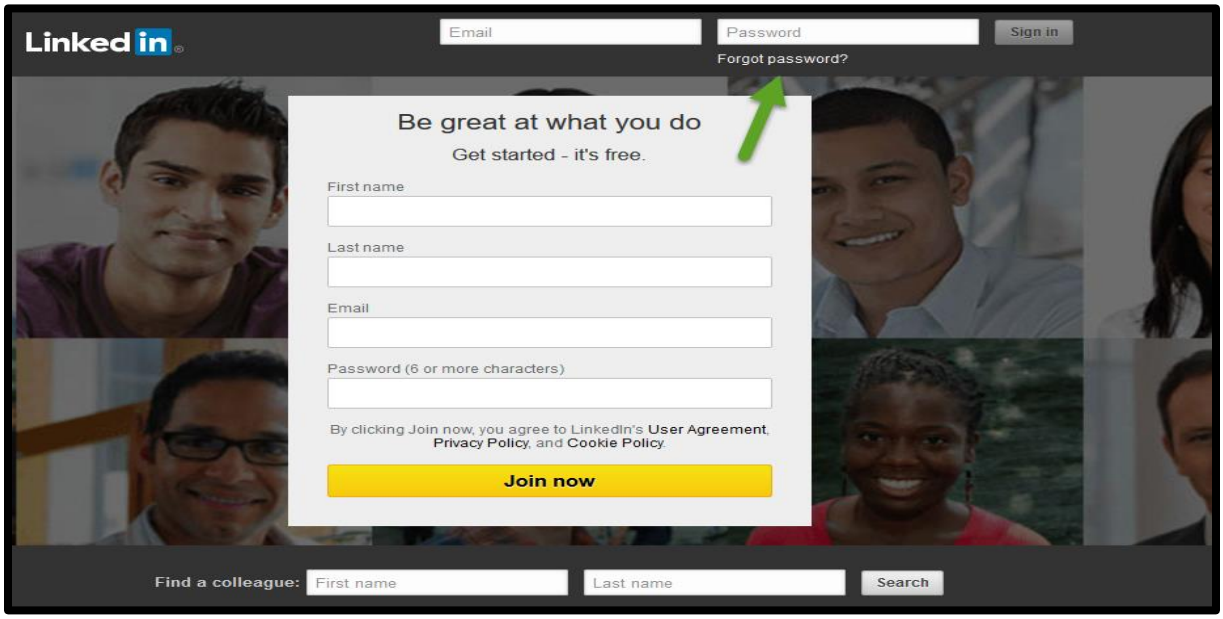


Figure 7.1 – LinkedIn's Home Screen (Forgot Password?)

```
<a class="link-forgot-password" href="https://www.linkedin.com/uas/request-password-reset?trk=uno-reg-guest-home-forgot-password">Forgot password?</a>
```

Figure 7.2 – HTML For The Forgot Password Hyperlink On LinkedIn's Home Page

Click The Hyperlink For Forgot Password (Link Text)

The following is code for clicking the Forgot Password hyperlink on LinkedIn's Home Page via linkText locator type:

```
1  package LinkedInHomePage;
2  import org.openqa.selenium.*;
5
6  public class ForgotPassword
7  {
8      WebDriver driver;
9
10     @BeforeTest
11     public void setUp() throws Exception
12     {
13         System.setProperty("webdriver.chrome.driver", "C:\\Users\\REX A JONES\\Downloads\\chromedriver_win32\\chromedriver.exe");
14         driver = new ChromeDriver();
15
16         driver.get("https://www.linkedin.com/");
17     }
18
19     @AfterTest
20     public void tearDown() throws Exception
21     {
22         driver.quit();
23     }
24
25     @Test
26     public void clickForgotPassword()
27     {
28         driver.findElement(By.linkText("Forgot password?")).click();
29     }
30 }
31
```

1. The **get()** method loads LinkedIn's Home page
2. The Link Text WebElement is found using **findElement()** and locator type **By.*linkText***
3. The hyperlink is clicked using **click()**
4. The **quit()** method closes the browser

Figure 7.3 – Perform Actions On The Forgot Password Hyperlink

- Line 8 "WebDriver driver" is the interface for driving the browser. Currently, the object reference variable "driver" points to nothing but will point to a Chrome Driver object in a subsequent line "driver = new ChromeDriver()".

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

- Line 13 tells Selenium where the executable file for Chrome driver is located via System.*setProperty*. The executable file operates like a bridge between Selenium and the browser. All browsers except Firefox require an executable file. Steps for downloading the executable file are located in Chapter 1 – Download Browser Drivers section. The following are parameters for System.*setProperty*:
  - **key** = webdriver.chrome.driver
  - **value** = C:\\Users\\REX A JONES\\Downloads\\chromedriver_win32\\chromedriver.exe
    Note**:** Value is the path location of the executable file

- Line 14 "driver = **new** ChromeDriver()" is an implementation of the WebDriver interface. The object reference variable "driver" is pointing to new ChromeDriver() which means testing is controlled on the Chrome browser.

- Line 16 "driver.get("https://www.linkedin.com/")" loads a new LinkedIn Home page in the current browser window

- Line 22 "driver.quit()" quits the driver instance and closes the open browser window

- Line 28 "driver.findElement(By.*linkText*("Forgot password?")).click()":
  - driver – WebDriver object reference variable that assist with finding a WebElement
  - findElement – a WebDriver method that finds the hyperlink WebElement "Forgot Password" on LinkedIn's Home page
  - (By.*linkText*("Forgot password")) – By and *linkText* are parameters of the findElement WebDriver method. By is an object which locate elements and linkText is the locator type. The Link Text locator type accepts a string parameter "Forgot password" which is the text name of the hyperlink
  - click() – clicks the Forgot Password hyperlink

Note: The submit() method can also be used to click the Forgot Password hyperlink.

The following is a screenshot of LinkedIn's Home page after clicking the Forgot Password hyperlink:

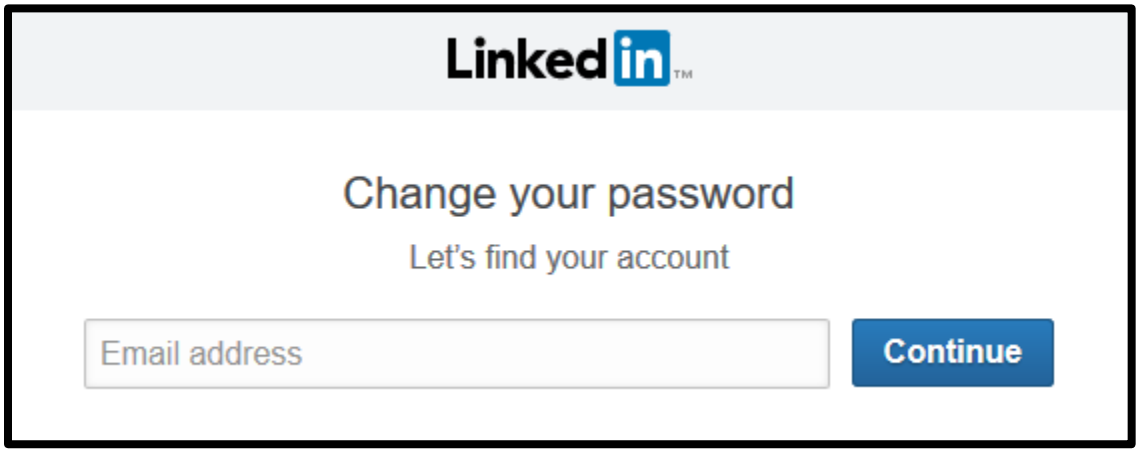


Figure 7.4 – Change Your Password

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Chapter 8
Find WebElement By Partial Link Text

Finding an element by Partial Link Text is carried out similar to Link Text. Both techniques only work for clicking hyperlinks. However, the difference between Partial Link Text and Link Text is hyperlink's text name. Part of the hyperlink's text name is suitable for the Partial Link Text locator type. For example, an automation engineer can enter "Password" and the link "Forgot Password" is clicked. The following is a screenshot of LinkedIn and its HTML markup tags for the Forgot Password hyperlink:

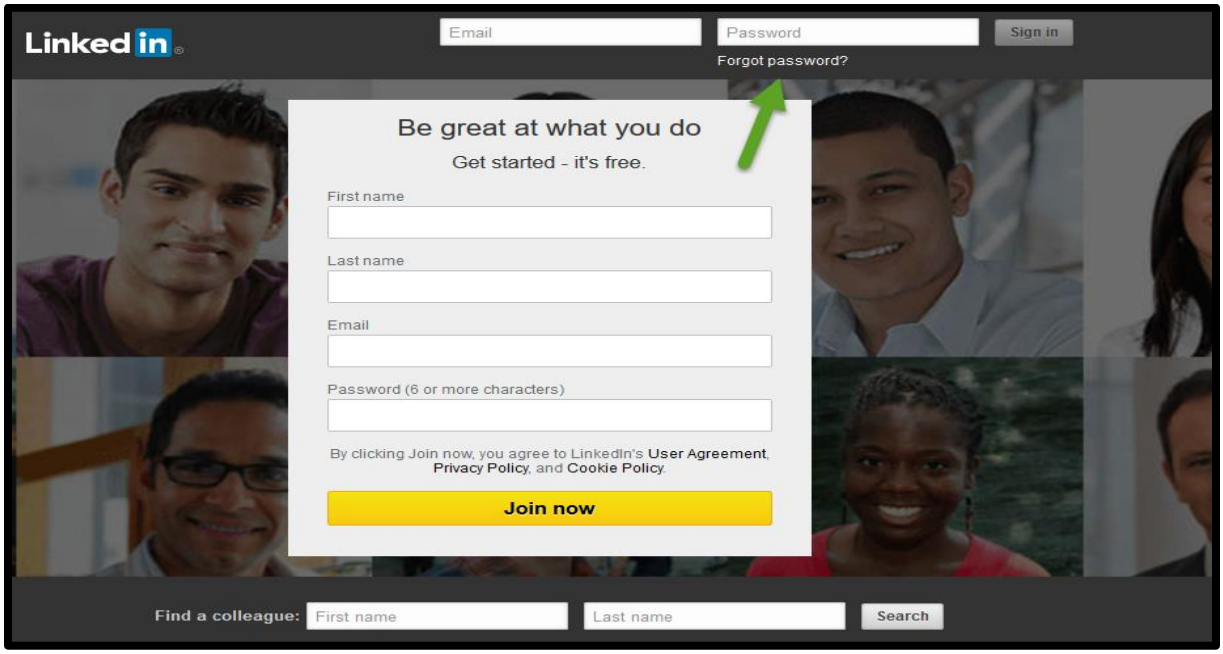


Figure 8.1 – LinkedIn's Home Screen (Forgot Password?)

```
<a class="link-forgot-password" href="https://www.linkedin.com/uas/request-password-reset?trk=uno-reg-guest-home-forgot-password">Forgot password?</a>
```

Figure 8.2 – HTML For The Forgot Password Hyperlink On LinkedIn's Home Page (2)

Click The Hyperlink For Forgot Password (Partial Link Text)

The following is code for clicking the Forgot Password hyperlink on LinkedIn's Home Page via partialLinkText locator type:

```
1  package LinkedInHomePage;
2  import org.openqa.selenium.*;
5
6  public class ForgotPassword2
7  {
8      WebDriver driver;
9
10     @BeforeTest
11     public void setUp() throws Exception
12     {
13         System.setProperty("webdriver.chrome.driver", "C:\\Users\\REX A JONES\\Downloads\\chromedriver_win32\\chromedriver.exe");
14         driver = new ChromeDriver();
15
16         driver.get("https://www.linkedin.com/");
17     }
18
19     @AfterTest
20     public void tearDown() throws Exception
21     {
22         driver.quit();
23     }
24
25     @Test
26     public void clickForgotPassword()
27     {
28         driver.findElement(By.partialLinkText("Forgot")).click();
29     }
30 }
```

1. The **get()** method loads LinkedIn's Home page
2. The Link Text WebElement is found using **findElement()** and locator type **By.partialLinkText**
3. The hyperlink is clicked using **click()**
4. The **quit()** method closes the browser

Figure 8.3 – Perform Actions On The Forgot Password Hyperlink

- Line 8 "WebDriver driver" is the interface for driving the browser. Currently, the object reference variable "driver" points to nothing but will point to a Chrome Driver object in a

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

subsequent line "driver = new ChromeDriver()".

- Line 13 tells Selenium where the executable file for Chrome driver is located via System.*setProperty*. The executable file operates like a bridge between Selenium and the browser. All browsers except Firefox require an executable file. Steps for downloading the executable file are located in Chapter 1 – Download Browser Drivers section. The following are parameters for System.*setProperty*:
  - **key** = webdriver.chrome.driver
  - **value** = C:\\Users\\REX A JONES\\Downloads\\chromedriver_win32\\chromedriver.exe
    Note**:** Value is the path location of the executable file

- Line 14 "driver = **new** ChromeDriver()" is an implementation of the WebDriver interface. The object reference variable "driver" is pointing to new ChromeDriver() which means testing is controlled on the Chrome browser.

- Line 16 "driver.get("https://www.linkedin.com/")" loads a new LinkedIn Home page in the current browser window

- Line 22 "driver.quit()" quits the driver instance and closes the open browser window

- Line 28 "driver.findElement(By.*partialLinkText*("Forgot")).click()":
  - driver – WebDriver object reference variable that assist with finding a WebElement
  - findElement – a WebDriver method that finds the hyperlink WebElement "Forgot Password" on LinkedIn's Home page
  - (By.*partialLinkText*("Forgot")) – By and *partialLinkText* are parameters of the findElement WebDriver method. By is an object which locate elements and partialLinkText is the locator type. The Partial Link Text locator type accepts a string parameter "Forgot" which is part of the hyperlink's text name..
  - click() – clicks the Forgot Password hyperlink

Note: The submit() method can also be used to click the Forgot Password hyperlink.

The following is a screenshot of LinkedIn's Home page after clicking the Forgot Password hyperlink:

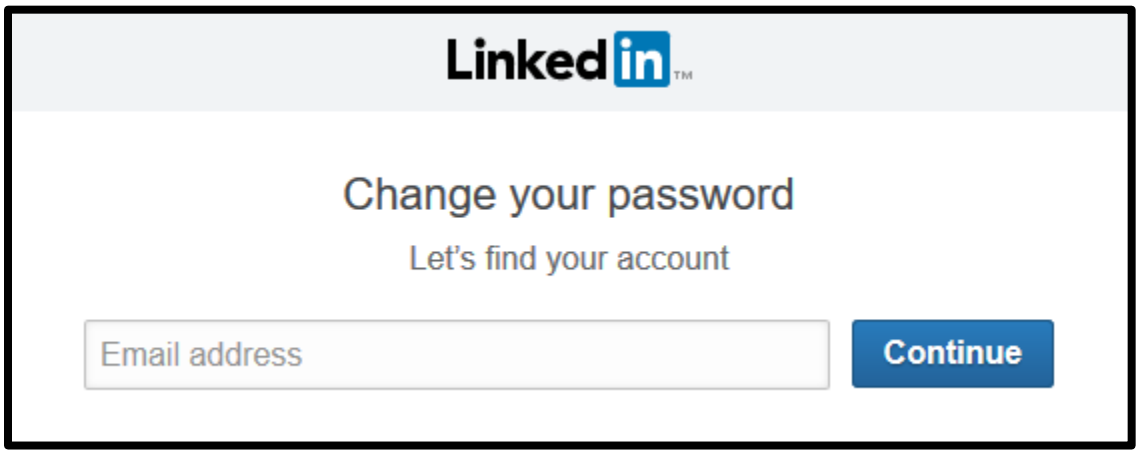


Figure 8.4 – Change Your Password

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Chapter 9
Find WebElement By Tag Name

In HTML, multiple elements share the same Tag Name. Therefore, finding an element by Tag Name is limited. It is best to use another locator type in conjunction with the Tag Name locator type to identify an element. The following is a screenshot of LinkedIn and its HTML markup tags for the Join Now button:

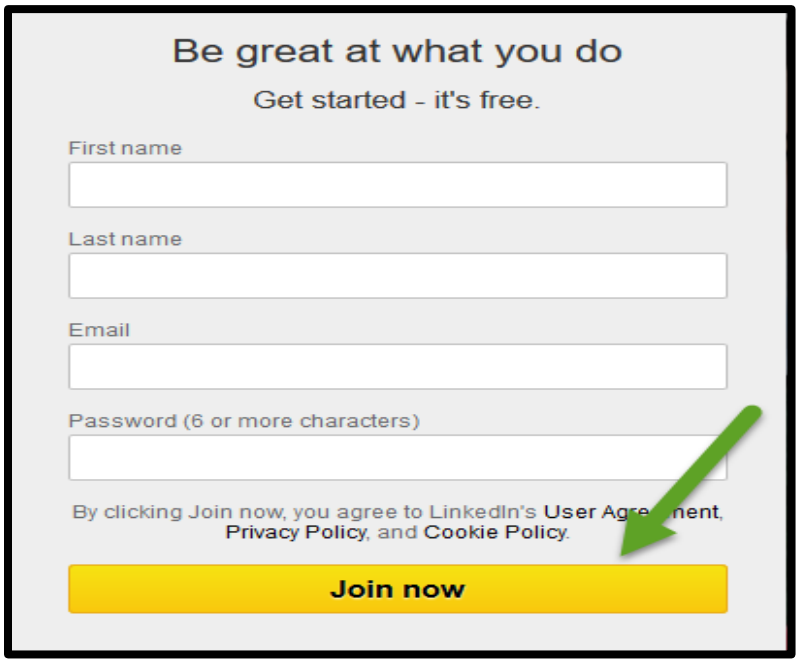


Figure 9.1 – LinkedIn's Join Now Button

```
<button class="btn btn-primary join-btn" data-position-join="right" type="submit">
    <span class="fill-v2">Join now</span>
</button>
```

Figure 9.2 – HTML For The Join Now Button On LinkedIn's Home Page

Click The Join Now Button

The following is code for clicking the Join Now button on LinkedIn's Home Page via tagname locator type:

```
1  package LinkedInHomePage;
2  import org.openqa.selenium.*;
5
6  public class JoinNow
7  {
8      WebDriver driver;
9
10     @BeforeTest
11     public void setUp() throws Exception
12     {
13         System.setProperty("webdriver.chrome.driver", "C:\\Users\\REX A JONES\\Downloads\\chromedriver_win32\\chromedriver.exe");
14         driver = new ChromeDriver();
15
16         driver.get("https://www.linkedin.com/");
17     }
18
19     @AfterTest
20     public void tearDown() throws Exception
21     {
22         driver.quit();
23     }
24
25     @Test
26     public void clickJoinNowButton ()
27     {
28         // This button can be clicked using submit() or click()
29         driver.findElement(By.tagName("button")).submit();
30     }
31 }
32
```

1

4

2 3

1. The **get()** method loads LinkedIn's Home page
2. The Button WebElement is found using **findElement()** and locator type **By.*tagName***
3. The button is clicked using **submit()**
4. The **quit()** method closes the browser

Figure 9.3 – Perform Actions On The Join Now Button

- Line 8 "WebDriver driver" is the interface for driving the browser. Currently, the object reference variable "driver" points to nothing but will point to a Chrome Driver object in a

subsequent line "driver = new ChromeDriver()".

- Line 13 tells Selenium where the executable file for Chrome driver is located via System.*setProperty*. The executable file operates like a bridge between Selenium and the browser. All browsers except Firefox require an executable file. Steps for downloading the executable file are located in Chapter 1 – Download Browser Drivers section. The following are parameters for System.*setProperty*:
  - **key** = webdriver.chrome.driver
  - **value** = C:\\Users\\REX A JONES\\Downloads\\chromedriver_win32\\chromedriver.exe
    Note**:** Value is the path location of the executable file

- Line 14 "driver = **new** ChromeDriver()" is an implementation of the WebDriver interface. The object reference variable "driver" is pointing to new ChromeDriver() which means testing is controlled on the Chrome browser.

- Line 16 "driver.get("https://www.linkedin.com/")" loads a new LinkedIn Home page in the current browser window

- Line 22 "driver.quit()" quits the driver instance and closes the open browser window

- Line 29 "driver.findElement(By.*tagName*("button")).submit()":
  - driver – WebDriver object reference variable that assist with finding a WebElement
  - findElement – a WebDriver method that finds the button WebElement "Join Now" on LinkedIn's Home page
  - (By.*tagName*("button")) – By and *tagName* are parameters of the findElement WebDriver method. By is an object which locate elements and tagName is the locator type. The Tag Name locator type accepts a string parameter "button" which is the tag name for the Join Now button.
  - submit() – clicks the Join Now button

Note: The click() method can also be used to click the Join Now button. There is only one button on LinkedIn's Home page. Therefore, the Tag Name locator type is used by itself because multiple elements do not share the same Tag Name "button".

Chapter 9
Find WebElement By Tag Name (Part 1) Selenium WebDriver

The following is a screenshot of LinkedIn's Home page after clicking the Join Now button without entering required fields:

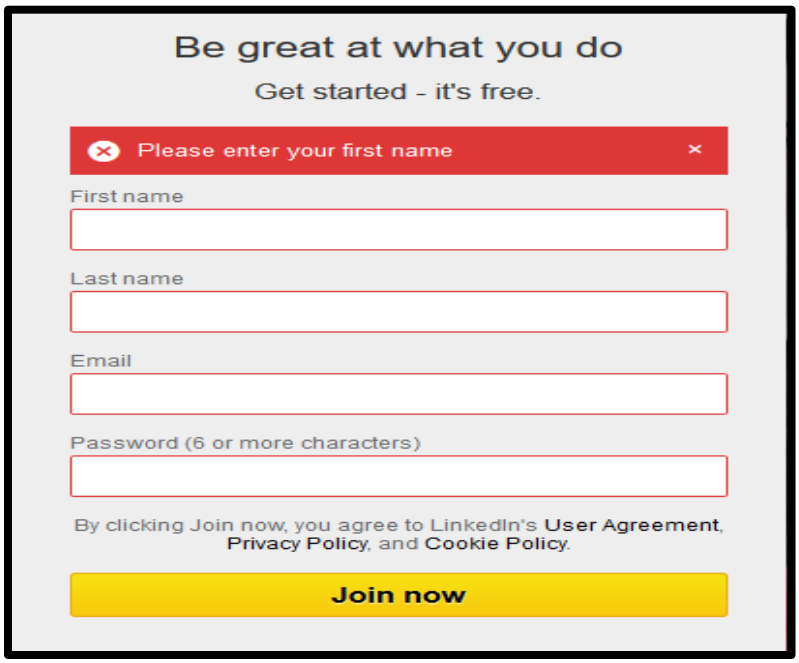


Figure 9.4 – Click The Join Now Button Without Entering Required Fields

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Chapter 10
Find WebElement By Class

Finding a WebElement by Class is unique from most locator types. A specific WebElement may contain multiple class attributes. On the other hand, there may be multiple WebElements within the same class attribute. If the latter scenario occurs then the first WebElement will be returned if the WebElement is not identified with another locator type. The following is a screenshot of LinkedIn and its HTML markup tags for the Email text box, Password text box, and Sign In button:

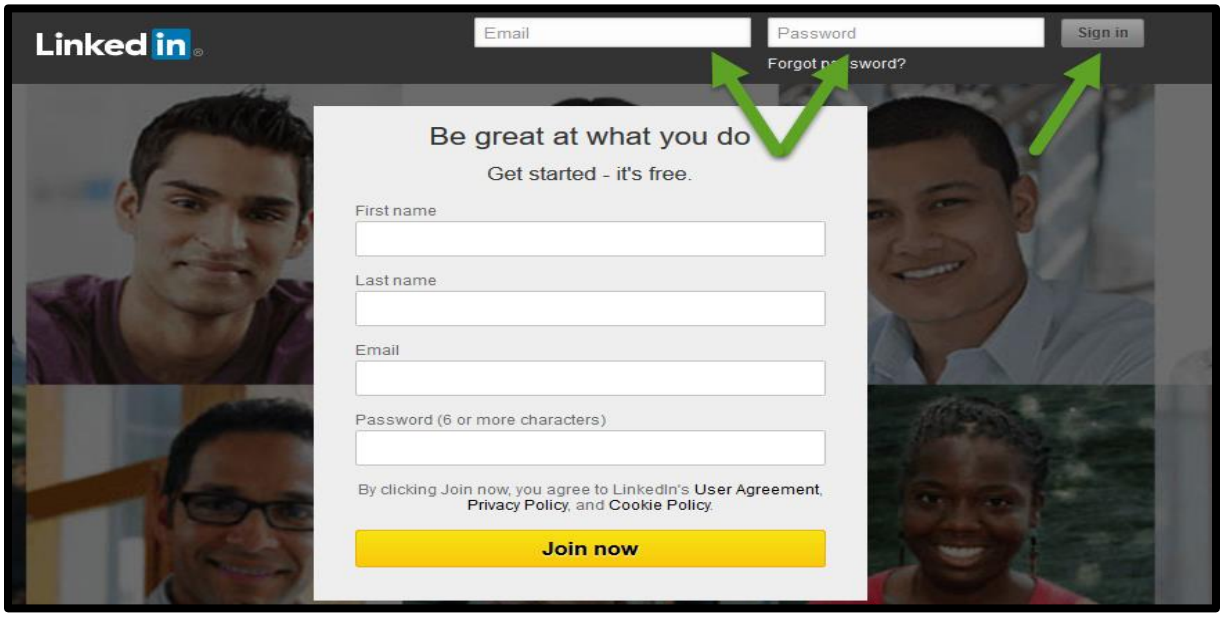


Figure 10.1 – LinkedIn's Email Text Field, Password Text Field, And Sign In Button

```
<form class="login-form" data-autologin="true" method="POST" action="https://www.linkedin.com/uas/login-submit">
    <label for="login-email">Email</label>
    <input id="login-email" type="text" autofocus="autofocus" placeholder="Email" name="session_key">
    <label for="login-password">Password</label>
    <input id="login-password" type="password" placeholder="Password" aria-required="true" name="session_password">
    <input type="hidden" value="false" name="isJsEnabled">
    <input id="loginCsrfParam-login" type="hidden" value="ffaa5956-19fc-4026-86b2-e367e2f2d8ae" name="loginCsrfParam">
    <input id="sourceAlias-login" type="hidden" value="0_7r5yezRXCiA_H0CRD8sf6DhOjTKUNps5xGTqeX8EEoi" name="sourceAlias">
    <input type="submit" value="Sign in" name="submit">
```

Figure 10.2 – HTML For The Email Text Field, Password Text Field, And Sign In Button

Enter Email, Password, and Click The Sign In Button

The following is code for entering an email, password, and clicking the Sign In button on LinkedIn's Home Page via classname locator type:

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

```java
package LinkedInHomePage;
import org.openqa.selenium.*;

public class CaptureErrorMessage
{
    WebDriver driver;

    @BeforeTest
    public void setUp() throws Exception
    {
        System.setProperty("webdriver.chrome.driver", "C:\\Users\\REX A JONES\\Downloads\\chromedriver_win32\\chromedriver.exe");
        driver = new ChromeDriver();

        driver.get("https://www.linkedin.com/");
    }

    @AfterTest
    public void tearDown() throws Exception
    {
        driver.quit();
    }

    @Test
    public void captureSignInErrorMessage()
    {

        WebElement parentSignIn = driver.findElement(By.className("login-form"));

        WebElement childEmail = parentSignIn.findElement(By.id("login-email"));
        WebElement childPassword = parentSignIn.findElement(By.id("login-password"));
        WebElement childSignInButton = parentSignIn.findElement(By.name("submit"));

        // Enter Email and Password then Click the Sign In button
        childEmail.sendKeys("Rex.Jones@Test4Success.org");
        childPassword.sendKeys("34Tester");
        childSignInButton.click();

        // Capture the error messages
        String firstSignInErrorMessage = driver.findElement(By.className("alert")).getText();
        System.out.println(firstSignInErrorMessage);
    }
}
```

1

5

2

3

4

1. The **get()** method loads LinkedIn's Home page

2. The text fields (Email & Password) and Sign In button are child WebElements found using **findElement()** but different locator types (**By.*id* & By.*name***). The parent WebElement was located using locator type **By.*className***.

3. Text entered for Email and Password using **sendKeys()** but the Sign In button is clicked using **click()**.

4. Capture the error message using **getText()** then prints the message

5. The **quit()** method closes the browser

Figure 10.3 – Perform Actions On Email, Password, And Sign In

Program Output:

```
There were one or more errors in your submission. Please correct the
marked fields below.
```

- Line 8 "WebDriver driver" is the interface for driving the browser. Currently, the object reference variable "driver" points to nothing but will point to a Chrome Driver object in a subsequent line "driver = new ChromeDriver()".

- Line 13 tells Selenium where the executable file for Chrome driver is located via System.*setProperty*. The executable file operates like a bridge between Selenium and the browser. All browsers except Firefox require an executable file. Steps for downloading the executable file are located in Chapter 1 – Download Browser Drivers section. The following are parameters for System.*setProperty*:
  - **key** = webdriver.chrome.driver
  - **value** = C:\\Users\\REX A JONES\\Downloads\\chromedriver_win32\\chromedriver.exe
    Note**:** Value is the path location of the executable file

- Line 14 "driver = **new** ChromeDriver()" is an implementation of the WebDriver interface. The object reference variable "driver" is pointing to new ChromeDriver() which means testing is controlled on the Chrome browser.

- Line 16 "driver.get("https://www.linkedin.com/")" loads a new LinkedIn Home page in the current browser window

- Line 22 "driver.quit()" quits the driver instance and closes the open browser window

- Line 29
  "WebElement parentSignIn = driver.findElement(By.*className*("login-form"))":
  - WebElement parentSignIn - creates an object reference variable called parentSignIn that refers or points to a WebElement. Subsequent lines for Email, Password, and Sign In will point to the same form WebElement via parentSignIn.
  - driver – WebDriver object reference variable that assist with finding the WebElement
  - findElement – a WebDriver method that finds the form WebElement on LinkedIn's Home page. The form includes an email text box, password text box, Sign In button, and a Forgot Password hyperlink
  - (By.*className*("login-form")) – By and *className* are parameters of the findElement WebDriver method. By is an object which locate elements and className is the locator type. The Class Name locator type accepts a string parameter "login-form" which is the value of the HTML class attribute.

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

- Line 31
  "WebElement childEmail = parentSignIn.findElement(By.*id*("login-email"))":
  - WebElement childEmail = creates an object reference variable called childEmail that refers or points to a WebElement
  - parentSignIn – object reference variable that assist with finding the WebElement by pointing to the same form WebElement from line 29
  - findElement – a WebDriver method that finds the text box WebElement "Email" on LinkedIn's Home page.
  - (By.*id*("login-email")) – By and *id* are parameters of the findElement WebDriver method. By is an object which locate elements and id is the locator type. The ID locator type accepts a string parameter "login-email" which is the value of the HTML id attribute.

- Line 32
  "WebElement childPassword = parentSignIn.findElement(By.*id*("login-password"))":
  - WebElement childPassword = creates an object reference variable called childPassword that refers or points to a WebElement
  - parentSignIn – object reference variable that assist with finding the WebElement by pointing to the same form WebElement from line 29
  - findElement – a WebDriver method that finds the text box WebElement "Password" on LinkedIn's Home page.
  - (By.*id*("login-password")) – By and *id* are parameters of the findElement WebDriver method. By is an object which locate elements and id is the locator type. The ID locator type accepts a string parameter "login-password" which is the value of the HTML id attribute.

- Line 33
  "WebElement childSignInButton = parentSignIn.findElement(By.*name*("submit"))":
  - WebElement childSignInButton = creates an object reference variable called childSignInButton that refers or points to a WebElement

- parentSignIn – object reference variable that assist with finding the WebElement by pointing to the same form WebElement from line 29
- findElement – a WebDriver method that finds the button WebElement "Sign In" on LinkedIn's Home page.
- (By.*name*("submit")) – By and *name* are parameters of the findElement WebDriver method. By is an object which locate elements and name is the locator type. The Name locator type accepts a string parameter "submit" which is the value of the HTML name attribute.

- Line 36 "childEmail.sendKeys("Rex.Jones@Test4Success.org)" types the text "Rex.Jones@Test4Success.org" in the Email text field via object reference childEmail

- Line 37 "childPassword.sendKeys("34Tester)" types the text "34Tester" in the Password text field via object reference childPassword

- Line 38 "childSignInButton.click()" clicks the Sign In button via object reference childSignInButton
  Note: The submit() method can also be used to click the Sign In button.

- Line 41
  "String firstSignInErrorMessage = driver.findElement(By.*className*("alert")).getText()"
  - String firstSignInErrorMessage = creates an object reference variable called firstSignInErrorMessage that refers or points to a string
  - driver – WebDriver object reference variable that assist with finding a WebElement
  - findElement – a WebDriver method that finds the WebElement string "error message" on LinkedIn's Sign In page
  - (By.*className*("alert")) – By and *className* are parameters of the findElement WebDriver method. By is an object which locate elements and className is the locator type. The Class Name locator type accepts a string parameter "alert" which is the value of the HTML class attribute

- getText() – gets the WebElement string

- Line 42 "System.***out***.println(firstSignInErrorMessage)" prints the WebElement string "error message". View the error message (Class's Program Output and/or Figure 13.4 – Sign In Page After Entering Invalid Credentials)

Note: In this example scenario, Email, Password, and Sign In contained unique attribute/values within HTML's source code. However, the Class locator type is beneficial if there are multiple WebElements with the same attribute/value. The WebElement with a duplicate attribute/value should be located first by finding the value for its class attribute followed by attributes ID, Name, etc. This explains why object reference "parentSignIn" was created for the form followed by Email "childEmail", Password "childPassword", and Sign In "childSignInButton".

Lines 29 – 38 could have been created without references. The following code also enters an email, password, and clicks the Sign In button:

```
driver.findElement(By.id("login-email")).sendKeys("Rex.Jones@Test4Success.org");
driver.findElement(By.id("login-password")).sendKeys("34Tester");
driver.findElement(By.name("submit")).click();
```

The following is a screenshot of LinkedIn's Sign In page after entering invalid credentials:

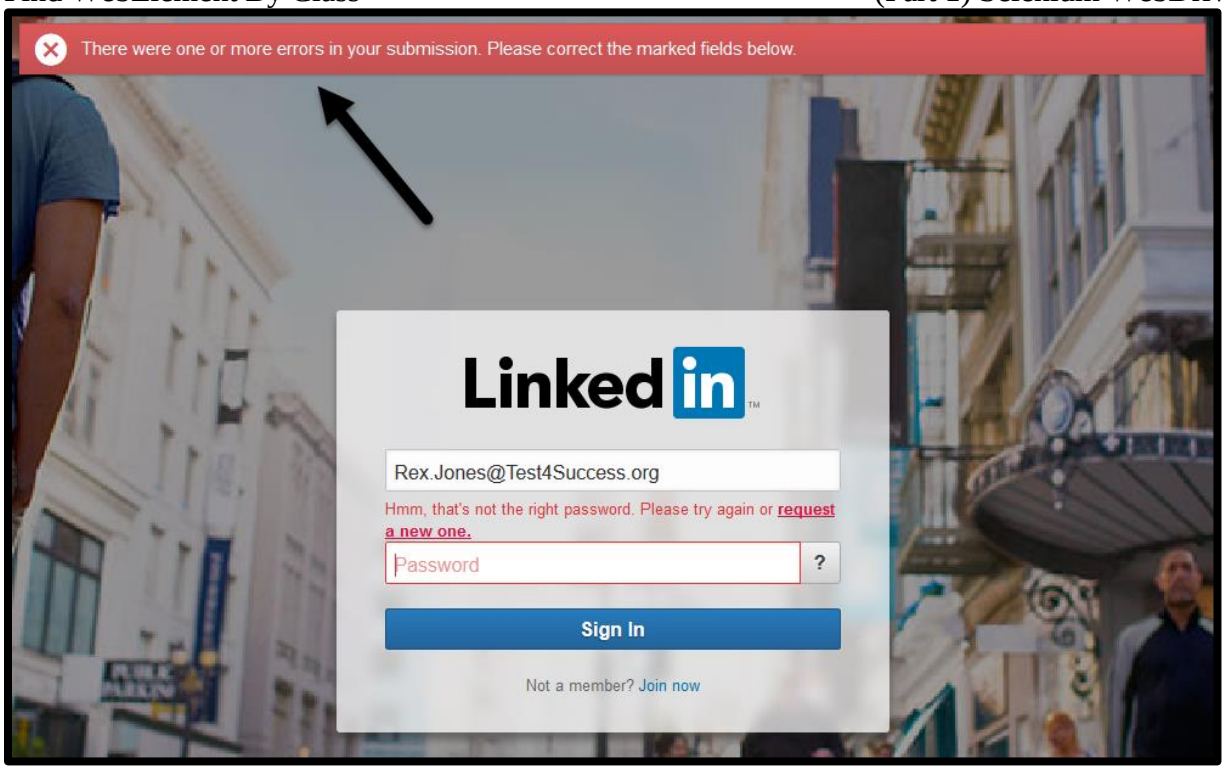


Figure 10.4 – Sign In Page After Entering Invalid Credentials

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Chapter 11
Browser Methods

The Browser Methods are a group of methods that perform an action on the browser. Each method is accessed by writing its name. The method's name is available after creating an object of WebDriver then writing the object's name followed by a dot operator. A parameter which is a value passed to the method is required for the get() method. However, a parameter is not required for all of the other Browser Methods. Methods that return a value will show it's return type "i.e., String" while methods that do not return a value will show void. The following is a screenshot and description of each Browser Method:

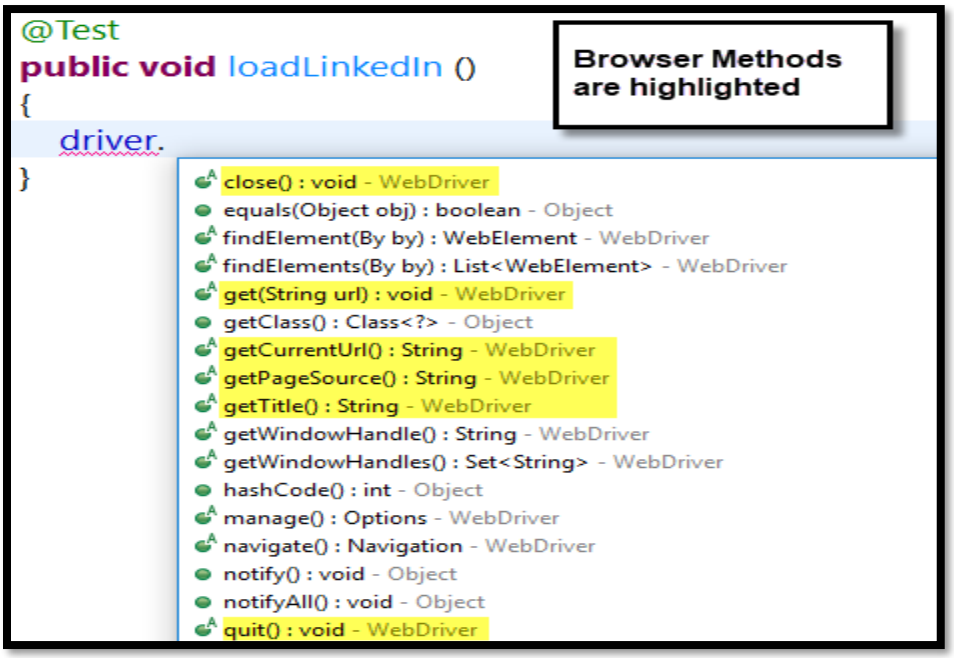


Figure 11.1 – Browser Methods After Writing Object And Dot Operator

Browser Method	Description
close()	Closes the current active window if there are multiple windows. The browser quits if only one window is active
get(String url)	Loads a new web page in the current browser window
getCurrentUrl()	Gets a string defining the current web page URL
getPageSource()	Gets the complete page source of the loaded web page.
getTitle()	Gets the current page title
quit()	Stops/Quits the driver instance and close all open browser windows

Figure 11.2 – Description Of Browser Methods

The following screenshot of get() method receiving a String parameter "https://www.linkedin.com" was also shown in Chapter 2 – WebDriver Objects and Methods. It is important to know the String parameter is a URL that must include https://. An exception shows up if the fully qualified URL is not provided as a parameter.

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

```
public class LinkedIn
{
   WebDriver driver;

   @BeforeTest
   public void setUp () throws Exception
   {
      driver = new ChromeDriver ();
      driver.manage().window().maximize();
      driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
   }

   @Test
   public void loadLinkedIn ()
   {
      driver.get("https://www.linkedin.com/");
   }
}
```

Get Method loads a new web page "LinkedIn"

Figure 11.3 – Get Method Loads A New Web Page

As mentioned, all of the other Browser Methods do not require a parameter. However, some of the methods such as getTitle() have a return type of String. The following is a screenshot of getTitle() receiving no parameters, returning the value to a String Data Type called title, and printing the title.

```
@Test
public void loadLinkedIn ()
{
   driver.get("https://www.linkedin.com/");

   String title = driver.getTitle();
   System.out.println(title);
}
```

Figure 11.4 – Browser Method / getTitle()

Browser Methods close() and quit() are similar to each other. As a result, there is confusion amongst automation engineers between both methods. The difference comes down to how they interact with the driver.

- close() – close the browser window but the driver remains open after executing a Test Script
- quit() – close the browser window and the driver closes after executing a Test Script

A good illustration is to view Task Manager's Detail tab to see how close() and quit() operate in the background. Here's a couple of screenshots of the close() method and Task Manager after loading LinkedIn, getting the title, then printing the title.

Note: One execution already occurred via Figure 11.4 to print the title. This is the second execution.

```
@Test
public void loadLinkedIn ()
{
   driver.get("https://www.linkedin.com/");

   String title = driver.getTitle();
   System.out.println(title);
}

@AfterTest
public void tearDown ()
{
   driver.close();
}
```

Figure 11.5 – close() Method

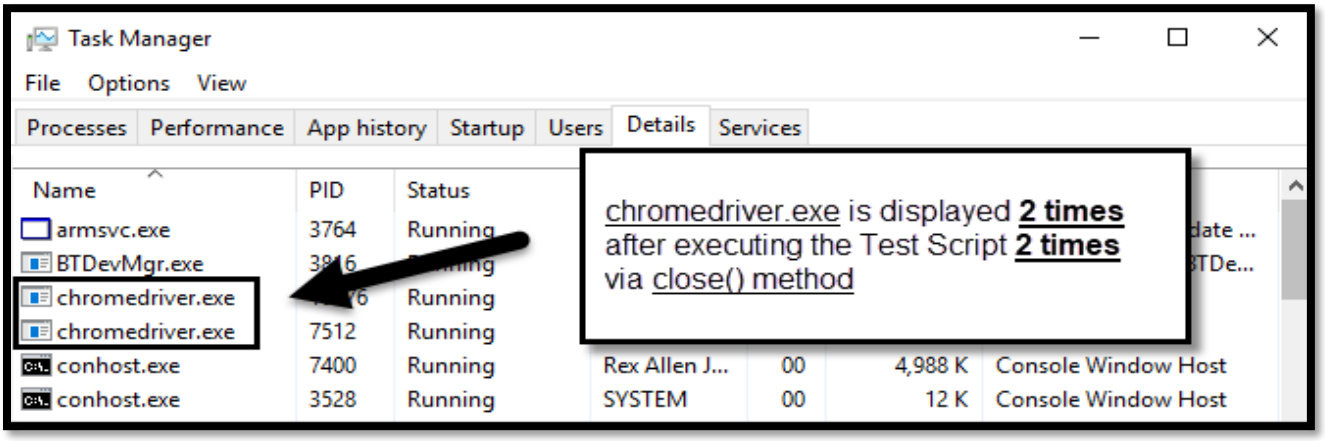


Figure 11.6 – chromedriver.exe After Executing Test Script via close() Method

Here's a couple of screenshots of the quit() method and Task Manager after loading LinkedIn, getting the title, then printing the title.

Note: Two executions have already occurred via Figure 11.4 to print the title and after executing the same Test Script then utilizing close() method. This is the third execution.

```
@Test
public void loadLinkedIn ()
{
   driver.get("https://www.linkedin.com/");

   String title = driver.getTitle();
   System.out.println(title);
}

@AfterTest
public void tearDown ()
{
   driver.quit();
}
```

Figure 11.7 – quit() Method

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

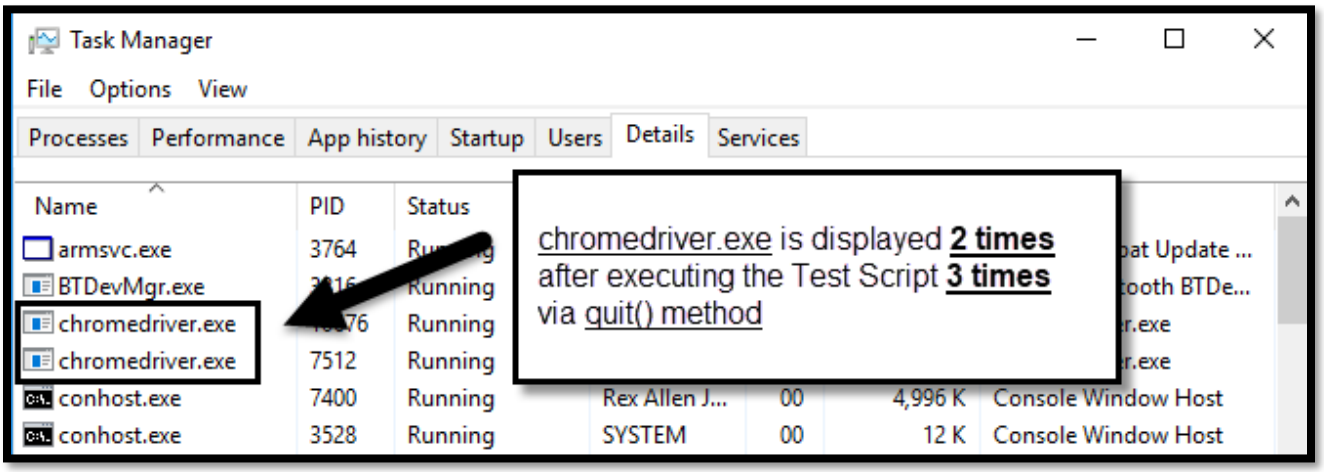


Figure 11.8 – chromedriver.exe After Executing Test Script via quit() Method

Chapter 12
Wait Methods

Wait Methods are a group of methods that pause between execution statements. They are beneficial for AJAX applications. AJAX stands for **A**synchronous **J**avaScript **a**nd **X**ML which is a technique for designing asynchronous applications. This type of application does not have to reload the web page while exchanging data with a server and updating parts of the web page. As a result, when a browser loads a web page, the WebElements may load at different time intervals. If a WebElement is not identified by a certain time, an exception will cause the Test Script to fail. Nevertheless, an automation engineer can avoid this problem using a Wait Method. There are several Wait Methods but the most utilized methods are Implicit Wait and Explicit Wait.

Implicit Wait

An Implicit Wait Method is used to implement a default waiting time for each Test Step within the entire Test Script. While waiting, WebDriver continues polling the Document Object Model (DOM) for the WebElement. If the WebElement is located before the default time then the next Test Step is executed. However, if the WebElement is not located within the specified time then an exception shows up. The following screenshots display a description and syntax for Implicit Wait.

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

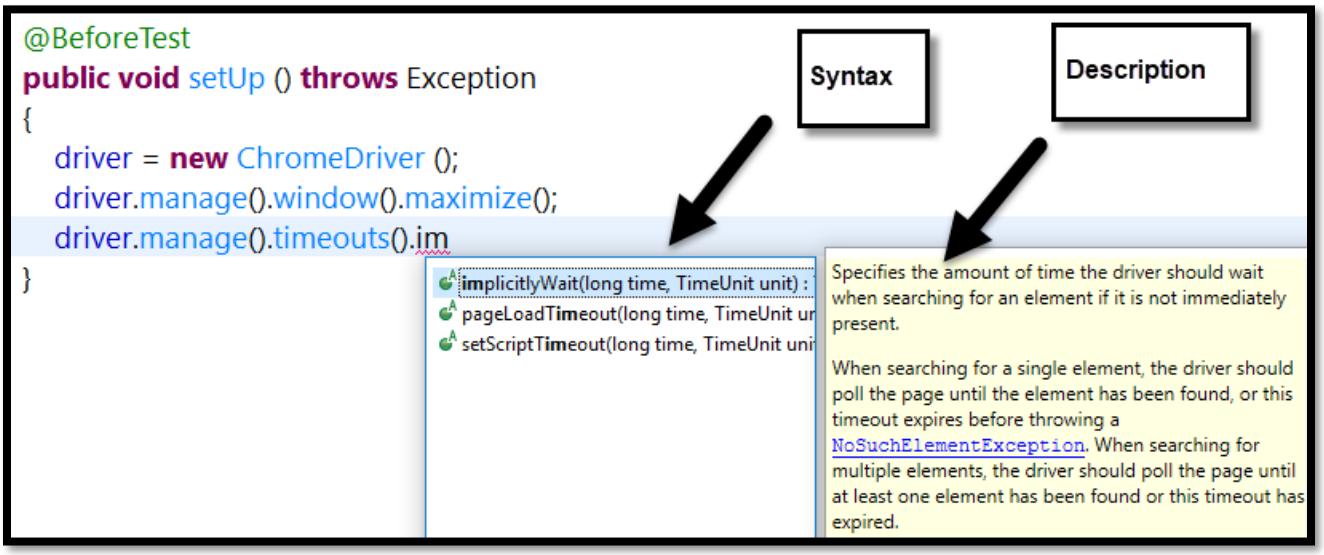


Figure 12.1 – Implicit Wait Syntax and Description

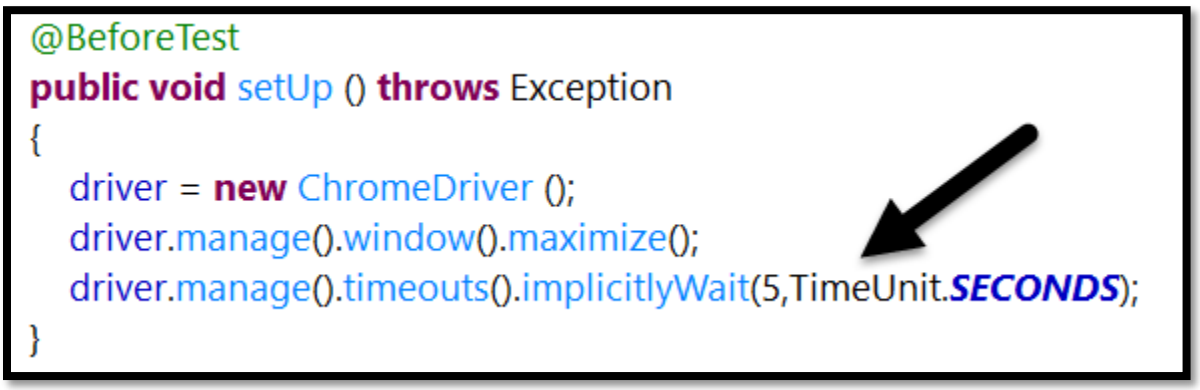


Figure 12.2 – Implicit Wait 5 Seconds

Implicit wait accepts the following 2 parameters:

- time - accepts the time as an integer value
- unit - accepts the time measurement in terms of SECONDS, NANOSECONDS, MINUTES, MILLISECONDS, MICROSECONDS, HOURS, and DAYS

Note: Usually, an Implicit Wait Method is provided prior to loading the web page within the setUp() method.

Explicit Wait

Explicit Wait Methods are used to interrupt execution until time has elapsed or an expected condition is met. That condition must be satisfied before proceeding to the next Test Step. Selenium WebDriver offers a WebDriverWait class and an ExpectedConditions class to assist Explicit Wait. It is important to know that Explicit Waits are limited to a specific WebElement.

The WebDriverWait class creates an object reference and use an instance of WebDriver. Also, a maximum number of seconds is added for execution inactivity. If the WebElement is not found then an exception is thrown. The following shows a description and syntax for WebDriverWait:

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

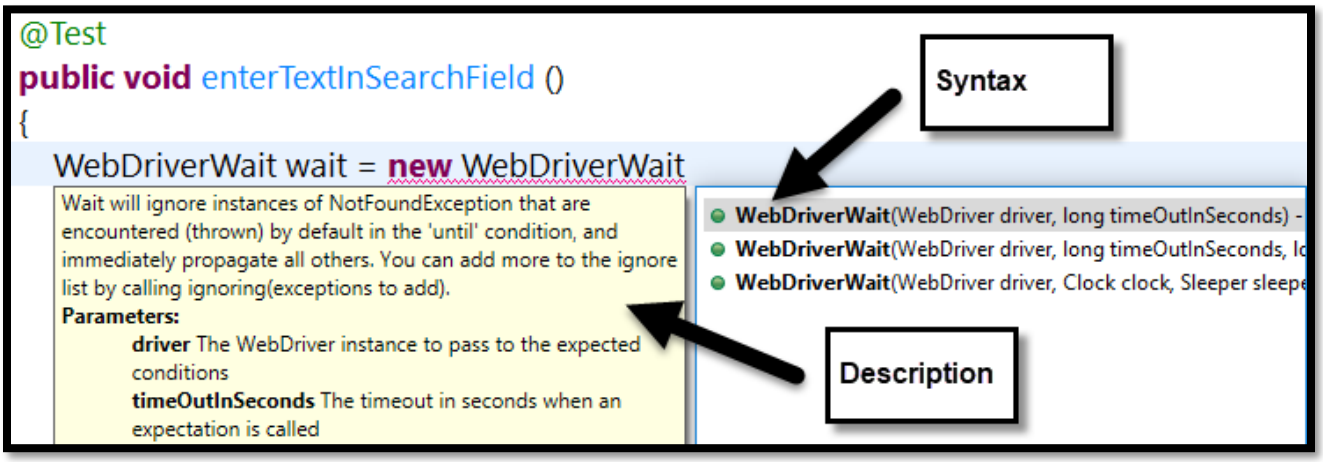


Figure 12.3 – WebDriverWait Syntax and Description

```
@Test
public void enterTextInSearchField ()
{
   WebDriverWait wait = new WebDriverWait(driver, 5);
```

Figure 12.4 – WebDriverWait – Time Out In 5 Seconds

In the above example, an object reference of wait is created for WebDriverWait while driver is the reference object for WebDriver. A maximum of 5 seconds is provided to locate the WebElement. Afterwards, an automation engineer needs to inform WebDriver to wait until an expected condition is met. The ExpectedConditions class supplies many methods for dealing with scenarios that may occur before executing the next Test Step. Here's a screenshot that displays some of the ExpectedConditions methods:

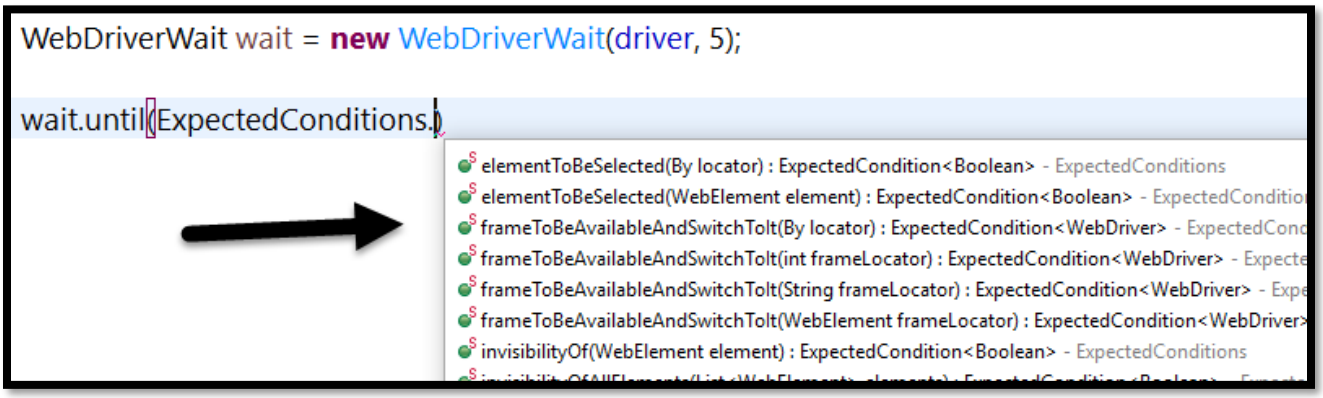


Figure 12.5 – Methods via ExpectedConditions Class

```
@Test
public void verifyCredentials ()
{
   WebDriverWait wait = new WebDriverWait(driver, 5);

   // Click Submit Button After It Becomes Enabled
   wait.until(ExpectedConditions.elementToBeClickable(By.id("submit")));

   // Next Test Step Is Executed After Above Condition Is Met
```

Figure 12.6 – ExpectedConditions Example

The ExepctedConditions class is accessed after writing the WebDriverWait reference “wait, dot operator, then until() method. Subsequently, many methods are available when writing dot after ExpectedConditions. The above statements wait for whichever happens first: wait for 5 seconds or

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

until the element is clickable. In alphabetical order, the following is a list of additional methods that is not shown in the screenshot:

1. alertIsPresent()
2. elementSelectionStateToBe()
3. elementToBeClickable()
4. invisibilityOfTheElementLocated()
5. invisibilityOfElementWithText()
6. presenceOfAllElementsLocatedBy()
7. presenceOfElementLocated()
8. textToBePresentInElement()
9. textToBePresentInElementLocated()
10. textToBePresentInElementValue()
11. titleIs()
12. titleContains()
13. visibilityOf()
14. visibilityOfAllElements()
15. visibilityOfAllElementsLocatedBy()
16. visibilityOfElementLocated()

The following is a table summarizing Implicit and Explicit Wait:

Implicit Wait	Explicit Wait
Single code line	Multiple code lines
Used on all WebElements within Test Script	Used for a specific WebElement
ExpectedConditions is not required	ExpectedConditions is required

Chapter 13
Navigation Methods

The Navigation Methods are utilized for loading a web page, refreshing a web page, moving backwards in your browser's history, and moving forward in your browser's history. Each Navigation Method is displayed after writing navigate then dot operator. The following is a screenshot and description of Navigation Methods.

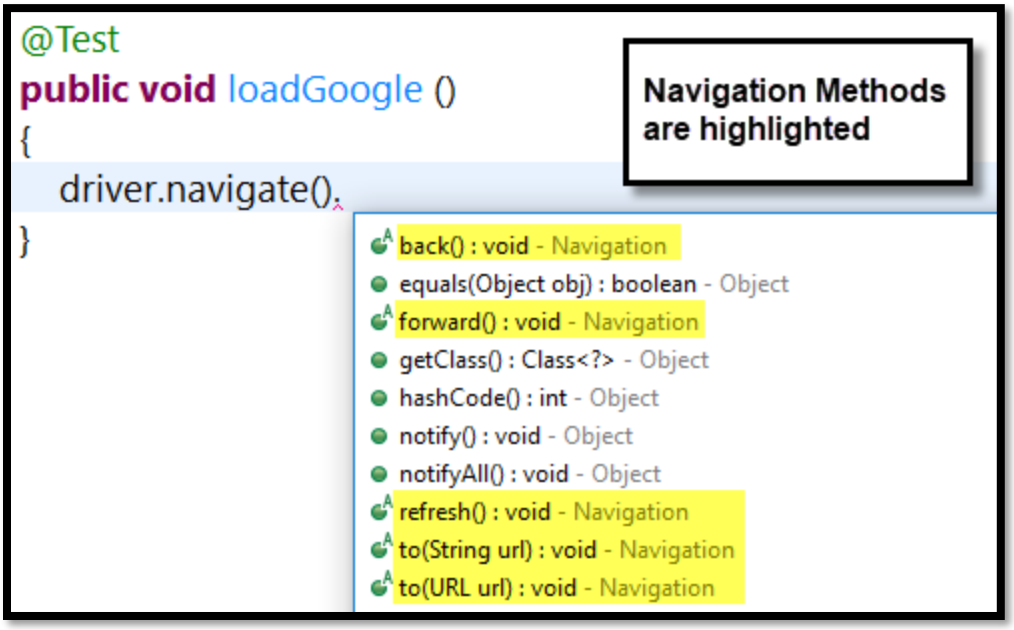


Figure 13.1 – Navigation Methods After Writing Navigate And Dot Operator

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Navigation Method	Description
back()	Moves back in the browser's history
forward()	Moves forward / next in the browser's history if not on the latest viewed page
refresh()	Refreshes the current page thereby reloading all WebElements
to(String url)	Loads a new web page
to(URL url)	Overloaded version of to(String url) method; loads a new web page

Figure 13.2 – Description of Navigation Methods

All of the Navigation Methods have a return type of void. However, only the navigate().to() methods require a parameter. As mentioned in Chapter 1 – Selenium WebDriver Method Categories, the get() method and navigate().to() method perform the same task of loading a new web page. In spite of their correspondence, it is a difference regarding how they maintain the browser's history.

- get() – loads a new web page and does not maintain the browser's history
- navigate().to() – loads a new web page and maintains the browser's history

Therefore, it is best to use the navigate().to() methods if a Test Script also moves back and forth via the browser's history. The following screenshot example verifies whether the back and forward buttons work after loading Google's Search page then refreshing Google's Search page:

```
@Test
public void loadGoogle ()
{
   driver.navigate().to("https://www.google.com");
   driver.navigate().refresh();
   driver.navigate().back();
   driver.navigate().forward();
}
```

Figure 13.3 – Example Of How To Write Each Navigation Method

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Chapter 14
WebElement Methods

According to Chapter 1 – Introduction To Selenium WebDriver and Chapter 2 – WebDriver and WebElements, a WebElement can be a button, text box, checkbox, drop-down menu, hyperlink, etc. In better words, a WebElement also known as element is anything visible on a web page. Therefore, WebElement Methods are a group of methods that perform an action on WebElements. The following table describes each WebElement Method that performs an action:

WebElement Method	Description
clear()	Clear or erase a value in a text field
click()	Clicks a link, button, checkbox, or radio button
findElement()	Find the first WebElement based on the locator type
findElements()	Find all elements within the current page based on the locator type
getAttribute()	Get the value of a specified WebElement attribute
getCssValue()	Returns the value of CSS properties
getLocation()	Returns the x and y coordinates with the top left corner of an element as the point
getSize()	Returns the width and height of a rendered element
getTagName()	Get the tag name of a specified WebElement

getText()	Get the visible innerText attribute of a WebElement
isDisplayed()	Verifies if a WebElement is present on a page
isEnabled()	Verifies if a WebElement is enabled on a page
isSelected()	Verifies if a WebElement is selected on a page
sendKeys()	Insert or type text in a text field
submit()	Operates like the click () method if the WebElement is a form or within a form

Figure 14.1 - Methods That Perform Actions On WebElements

Some of the WebElement Methods such as clear, click, getText, sendKeys, and submit were illustrated via different figures within the Locator Type chapters. The concepts remain the same for all elements:

1. First find the WebElement
2. Second perform an action on the WebElement

The most common WebElements (Text "Verbiage", Text Boxes, Buttons, Radio Buttons, and Hyperlinks "Links") were illustrated with actions in preceding chapters. The following common elements are illustrated with actions in this chapter:

- Checkboxes
- Drop-Down Menus

Actions Performed On Checkboxes

Actions performed on a checkbox is similar to a button, radio button, and hyperlink. WebElement Method click performs an action of clicking a checkbox. An automation engineer can also verify if

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

an element is displayed, enabled, or selected before/after performing an action. The following screenshots display syntax and description for click(), isDisplayed(), isEnabled(), and isSelected():

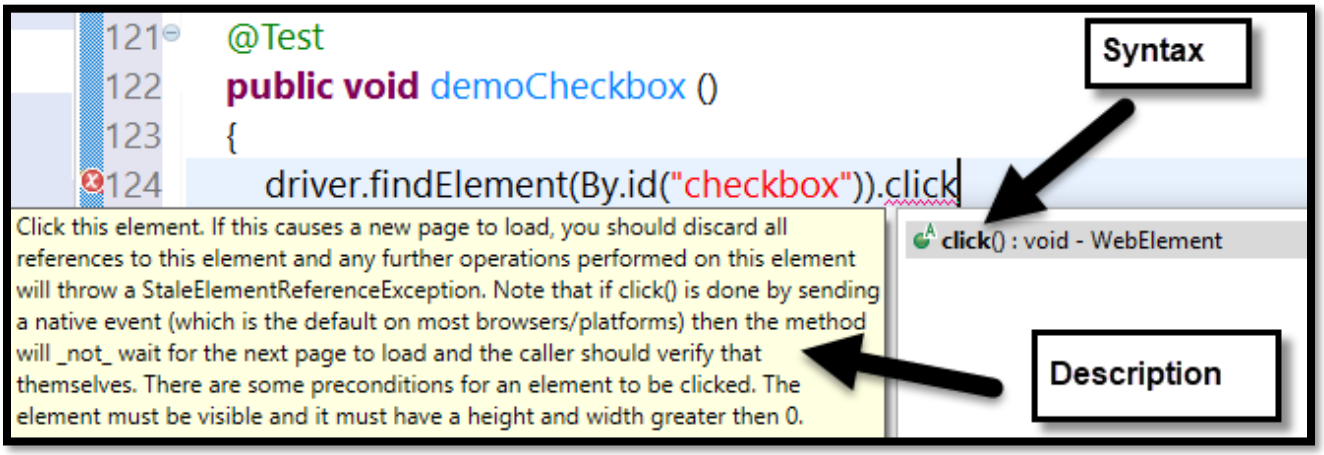


Figure 14.2 – Click Method Syntax and Description

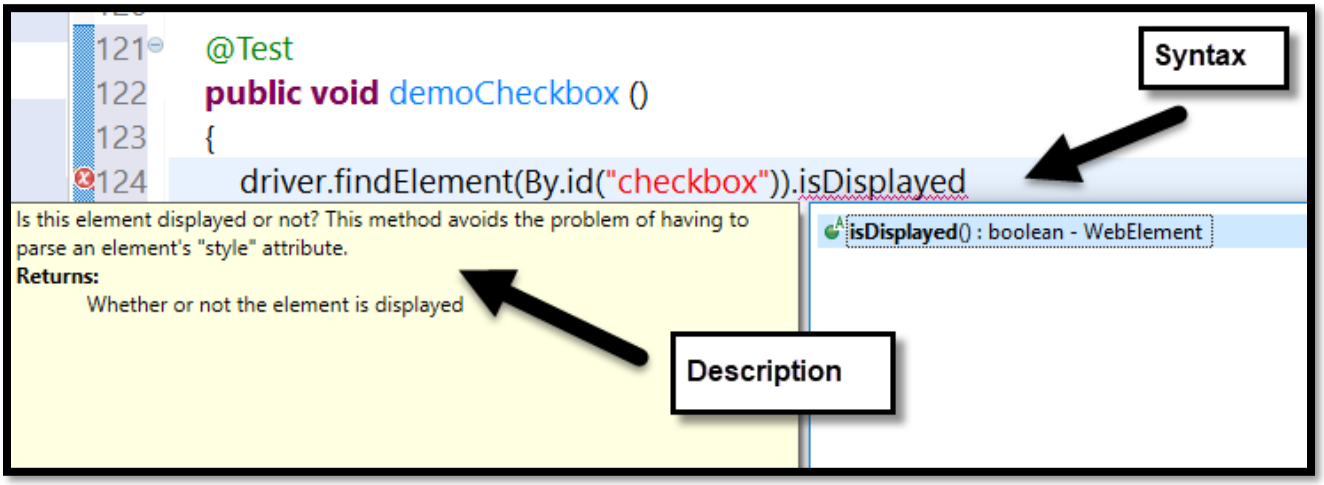


Figure 14.3 – isDisplayed Syntax and Description

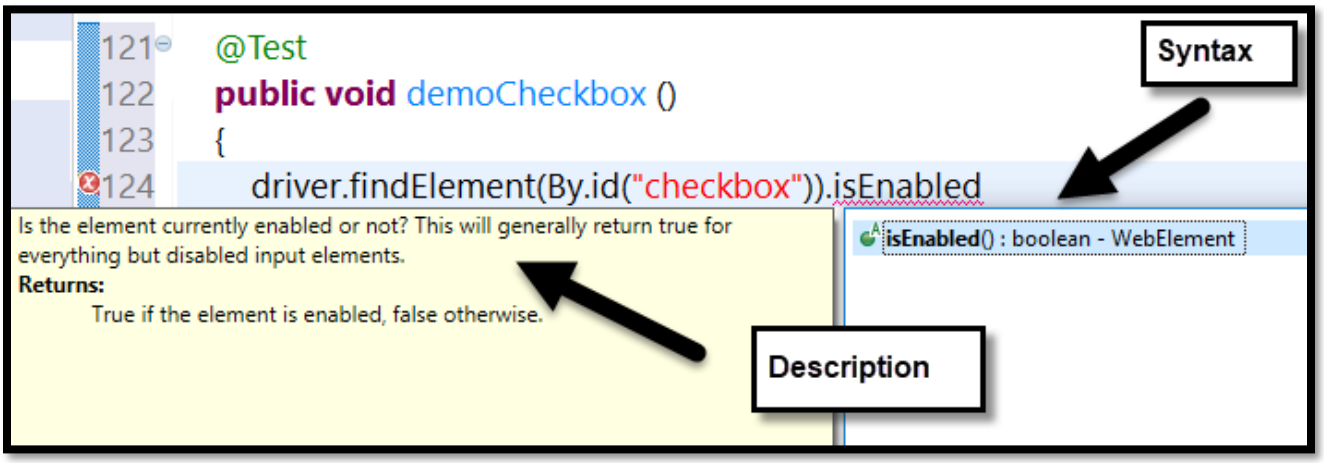


Figure 14.4 – isEnabled Syntax and Description

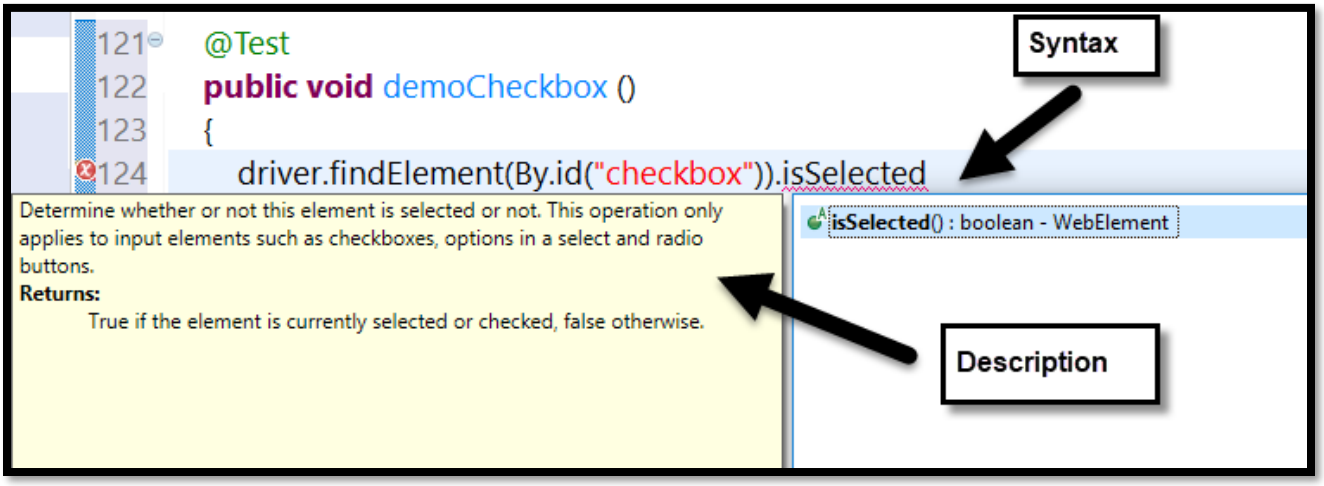


Figure 14.5 – isSelected Syntax and Description

Notice, isDisplayed(), isEnabled, and isSelected methods have a return value of boolean. Therefore, the value will be true or false. The following screenshots display a checkbox that is part of a web page and automation code for verifying if a specific checkbox is clicked after clicking the checkbox.

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

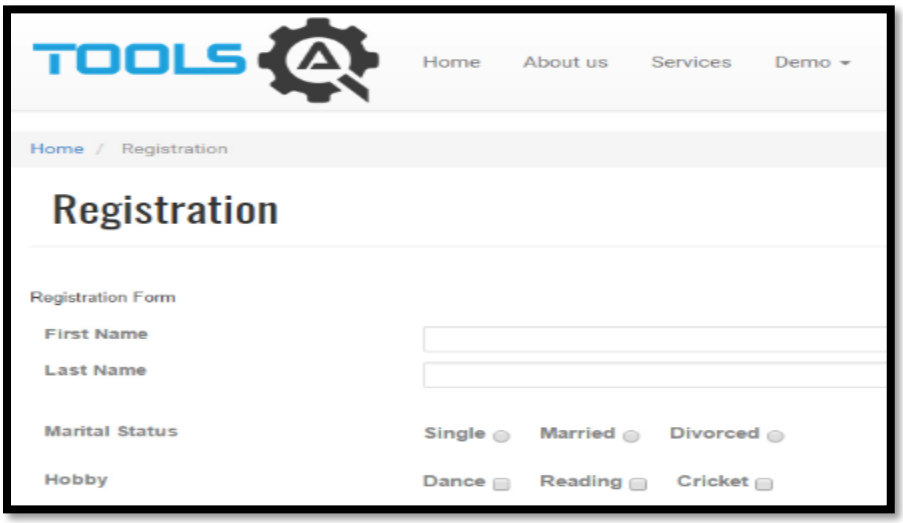


Figure 14.6 – Hobby Checkboxes via ToolsQA Web Page

```
36    @Test (priority = 1)
37    public void selectReadingHobby ()
38    {
39       List <WebElement> listCheckboxes = driver.findElements(By.name("checkbox_5[]"));
40
41       for (int i = 0; i < listCheckboxes.size(); i++)
42       {
43          if (listCheckboxes.get(i).getAttribute("value").equals("reading"))
44          {
45             listCheckboxes.get(i).click();
46          }
47       }
48
49       String selectedHobby = "";
50
51       for (int i = 0; i < listCheckboxes.size(); i++)
52       {
53          if (listCheckboxes.get(i).isSelected())
54          {
55             selectedHobby = listCheckboxes.get(i).getAttribute("value");
56          }
57       }
58       Assert.assertEquals(selectedHobby, "reading", "The Actual And Expected Results DO NOT Match");
59    }
```

Figure 14.7 – Example Code For Clicking A Checkbox

There are 3 checkboxes at the bottom of Figure 14.6 labeled Dance, Reading, and Cricket. The goal of automation code via Figure 14.7 is to click the Reading checkbox. Line 39 collects all 3 checkboxes via List <WebElement> listCheckboxes while Lines 41 – 47 cycle through each checkbox until Reading is clicked at Line 45. Lines 51 – 57 is a for loop and if statement that checks to see which checkbox is selected at Line 53. Finally, Line 58 is a TestNG Assertion that verifies whether the selected Hobby is Reading. Additional information regarding TestNG Assertions can be found in Getting Started With TestNG (A Java Test Framework): https://tinyurl.com/TestNG-Getting-Started

How To Select A Value From Drop-Down Menu

Performing actions on drop-down menus are different from all other WebElements. An automation engineer must still find the drop-down menu element and perform an action on the element. However, performing an action requires a Select class to select or deselect values from the drop-down menu. The following are screenshots for Select package and methods for Select class:

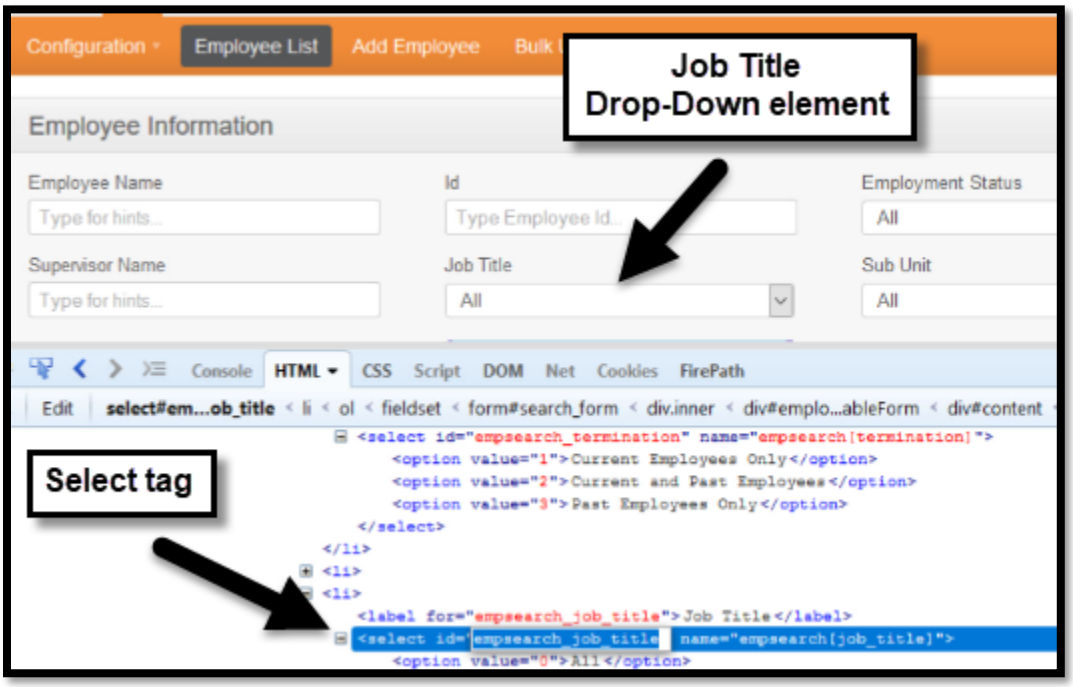


Figure 14.8 – Job Title Drop-Down Menu via OrangeHRM Web Page

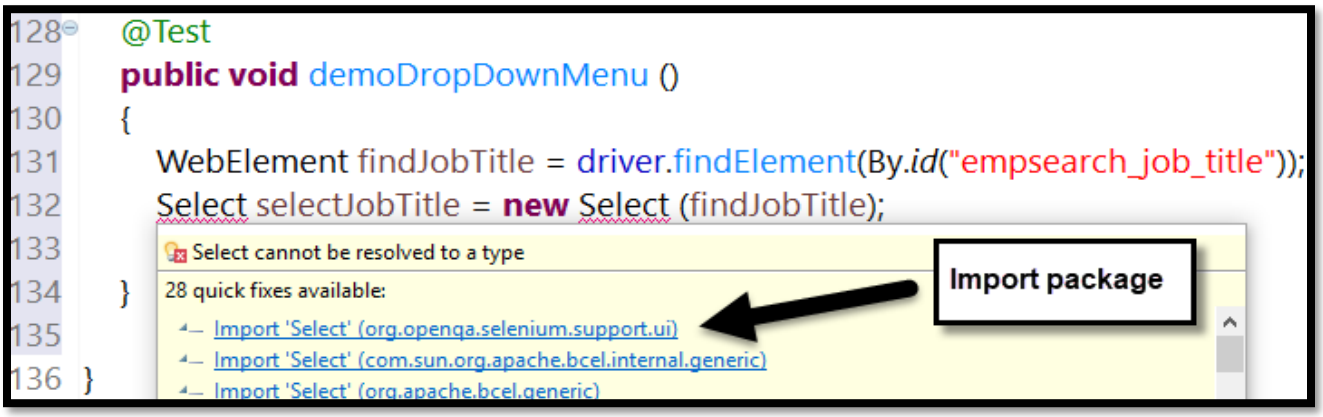


Figure 14.9 – Import Select Package / org.openqa.selenium.support.ui

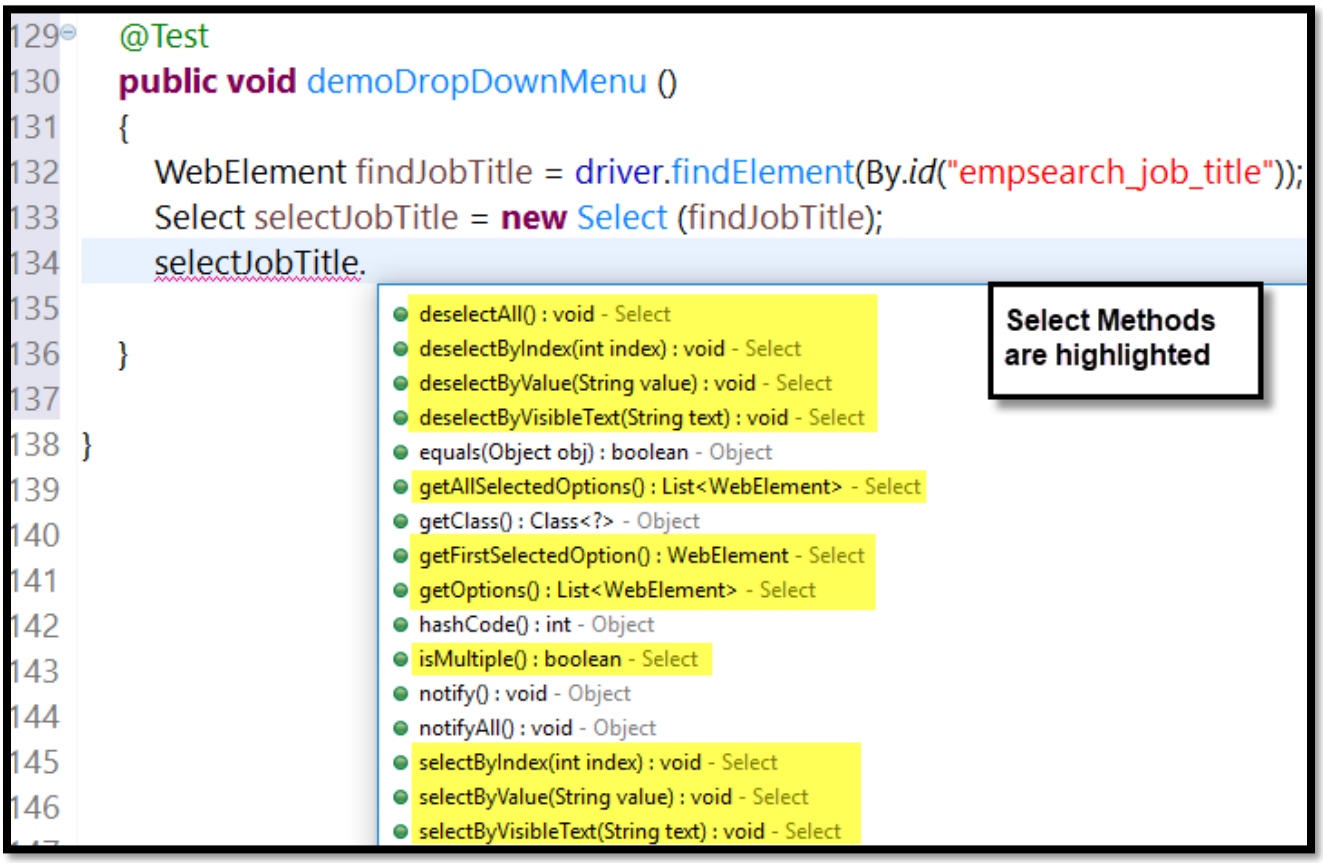


Figure 14.10 – Select Methods After Writing Select Object Reference And Dot Operator

Notice in Figure 14.9, the statement on Line 131 finds a WebElement like any other WebElement. In spite of that statement, the next statement on Line 132 separates a drop-down menu from other WebElements. A Select class must be imported to help Selenium WebDriver select or deselect a value. The Select class is an imitation of a select tag <select> within HTML illustrated via Figure 14.8. Figure 14.10 displays several methods for Select. The following screenshots display one of the ways for selecting a value from a drop-down menu:

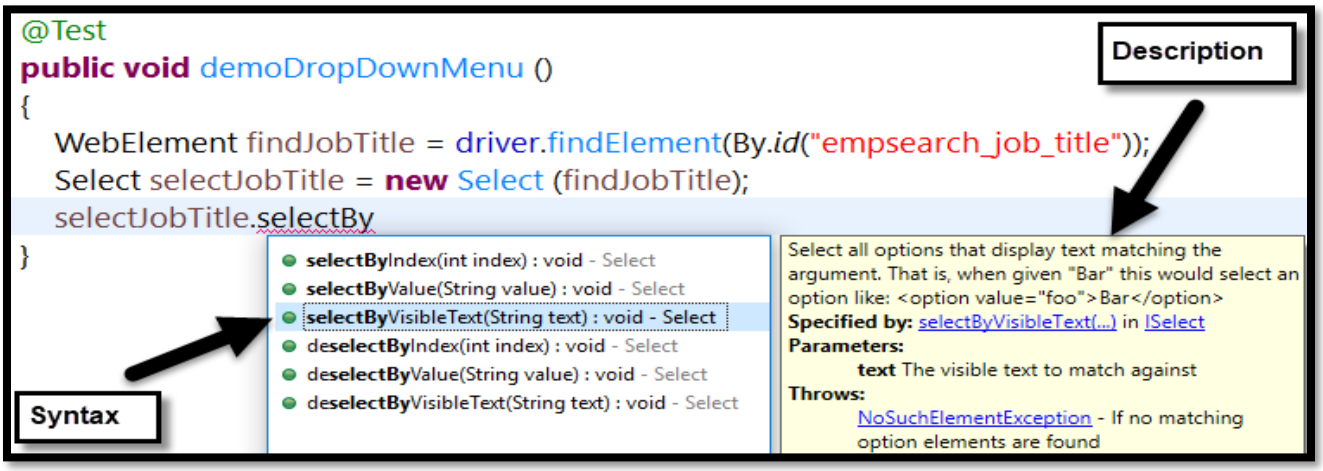


Figure 14.11 – selectByVisibleText Syntax and Description

```
129    @Test
130    public void demoDropDownMenu ()
131    {
132       WebElement findJobTitle = driver.findElement(By.id("empsearch_job_title"));
133       Select selectJobTitle = new Select (findJobTitle);
134       selectJobTitle.selectByVisibleText("QA Engineer");
```

Figure 14.12 – Example Code For Selecting An Option By selectByVisibleText

Line 134 illustrates how to select a value “QA Engineer” from a drop-down menu via selectByVisibleText() method. Values visible within the drop-down menu are utilized for this method.

Note: Deselecting a value via multi drop-down menu has the same concepts as selecting a value. Find an element then perform actions “select a value followed by deselecting a value”.

Chapter 15
Switch Methods

The Switch Methods are a group of methods used for switching to windows, frames, and alerts. An alert is better known as a pop-up box. Several methods are accessed via switchTo() which is a WebDriver.TargetLocator interface. The switchTo() method switches focus to the target element. Here is a screenshot that list each Switch Method:

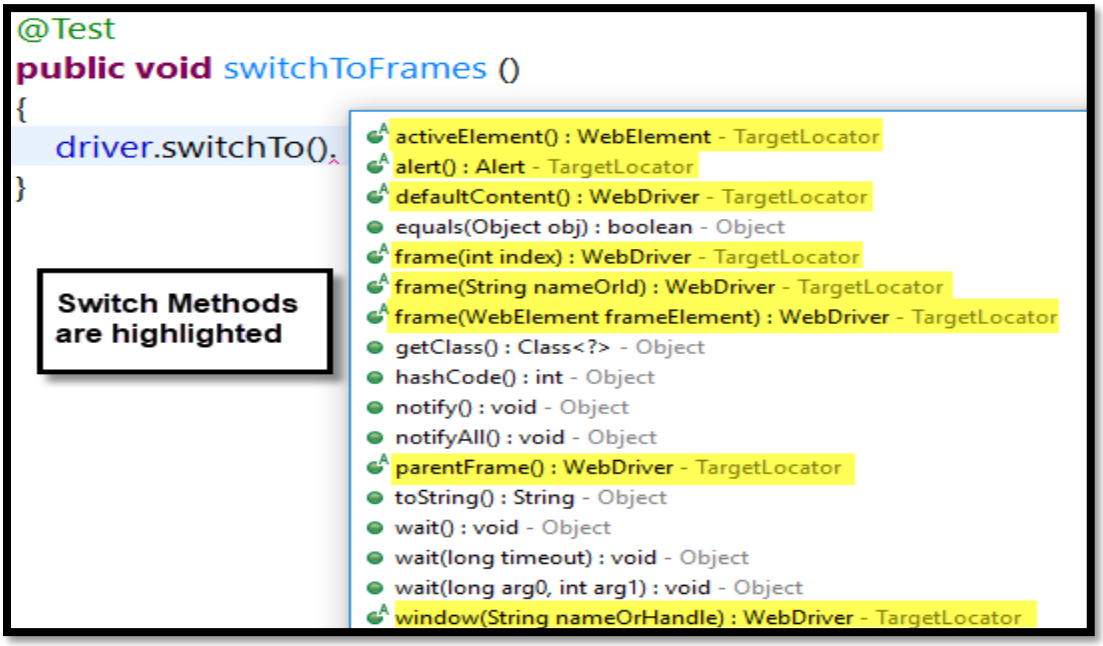


Figure 15.1 switchTo() Methods

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Windows

Selenium WebDriver assigns an alphanumeric id called window handle to all windows. The id is unique and used to switch control between each window. There are 2 methods that get the window handle and 1 method that switches between windows:

1. getWindowHandle() – get the current window handle
2. getWindowHandles() – get a set of window handles
3. switchTo().window() – switch focus between windows

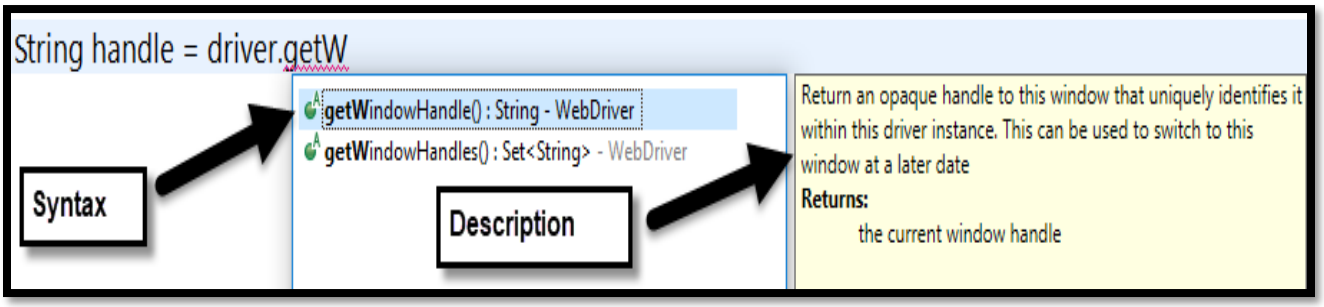


Figure 15.2 – getWindowHandle Syntax and Description

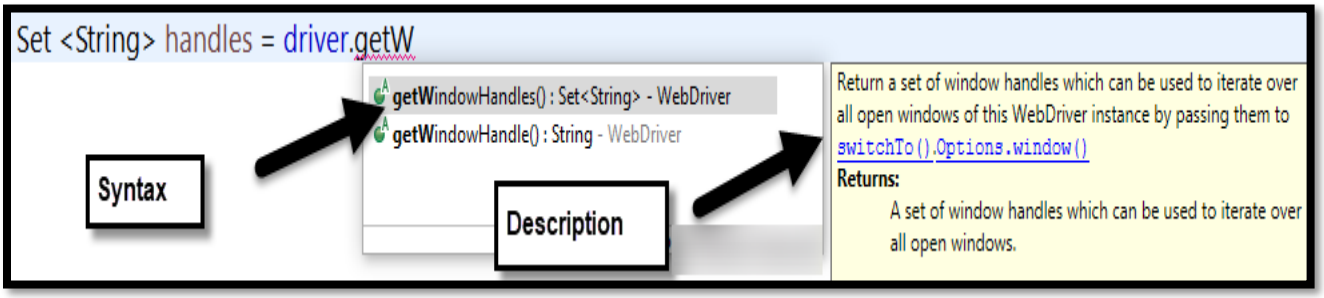


Figure 15.3 – getWindowHandles Syntax and Description

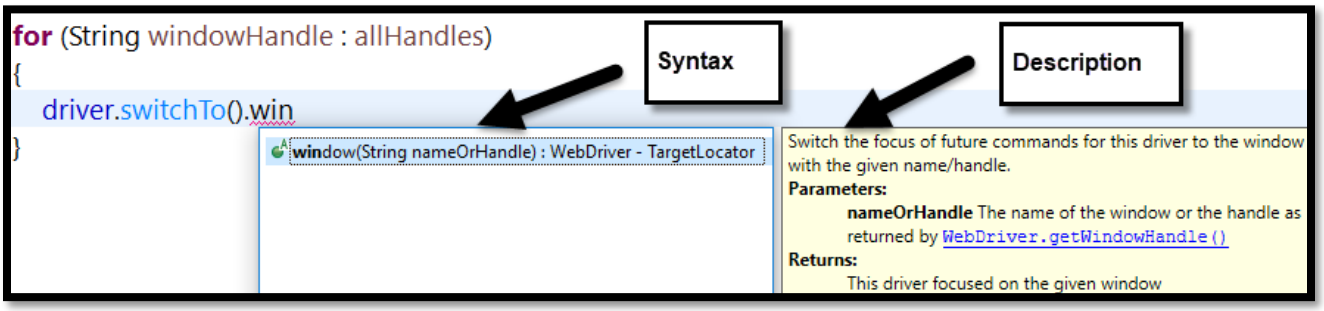


Figure 15.4 – switchTo().window() Syntax and Description

```
@Test
public void getWindowHandles ()
{
   // Get Window Handle
   String handle = driver.getWindowHandle();

   // Get Window Handles
   Set <String> allHandles = driver.getWindowHandles();

   // Switch To Window
   for (String windowHandle : allHandles)
   {
      driver.switchTo().window(windowHandle);
   }
}
```

Figure 15.5 – Example of getWindowHandle, getWindowHandles, and switchTo().window()

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

```
How Many Windows Are Open Before Loading The Web Page? 1
      Main ID:      CDwindow-bfb50a6a-351f-4d77-abf7-8901af879278

How Many Windows Are Open After Loading The Web Page? 2
      Window ID:    CDwindow-bfb50a6a-351f-4d77-abf7-8901af879278
      Window ID:    CDwindow-ab2f311c-bdbd-43f6-9bda-0e6eabaea615
```

Figure 15.6 –Window Handles

In Figure 15.5, a few Java statements are implemented to get each window handle and to switch between windows. The String class is used to assign the window handle to object reference "handle" while a set of window handles is assigned to object reference "allHandles". Java's for loop cycles through each window handle then assign the window handle to String windowHandle. Afterwards, the switchTo().window() method allows focus to be switched between windows.

Frames

Frames are a part of HTML tags. Within the HTML tag, there are 2 type of frames: frame tag <frame> and iframe tag <iframe>. The frame tags are used to divide a web page into different frames/windows while an iframe tag defines an inline frame. Inline frames embed additional web pages within the current web page. Sometimes it's difficult to detect an iframe by viewing the page. However, the following steps help identify an iframe.

1. Open Google Chrome browser
2. Go to a web page "i.e., Amazon"
3. Right click
4. Select Inspect
5. Select CTRL + F
6. Type //iframe
7. Click Enter key

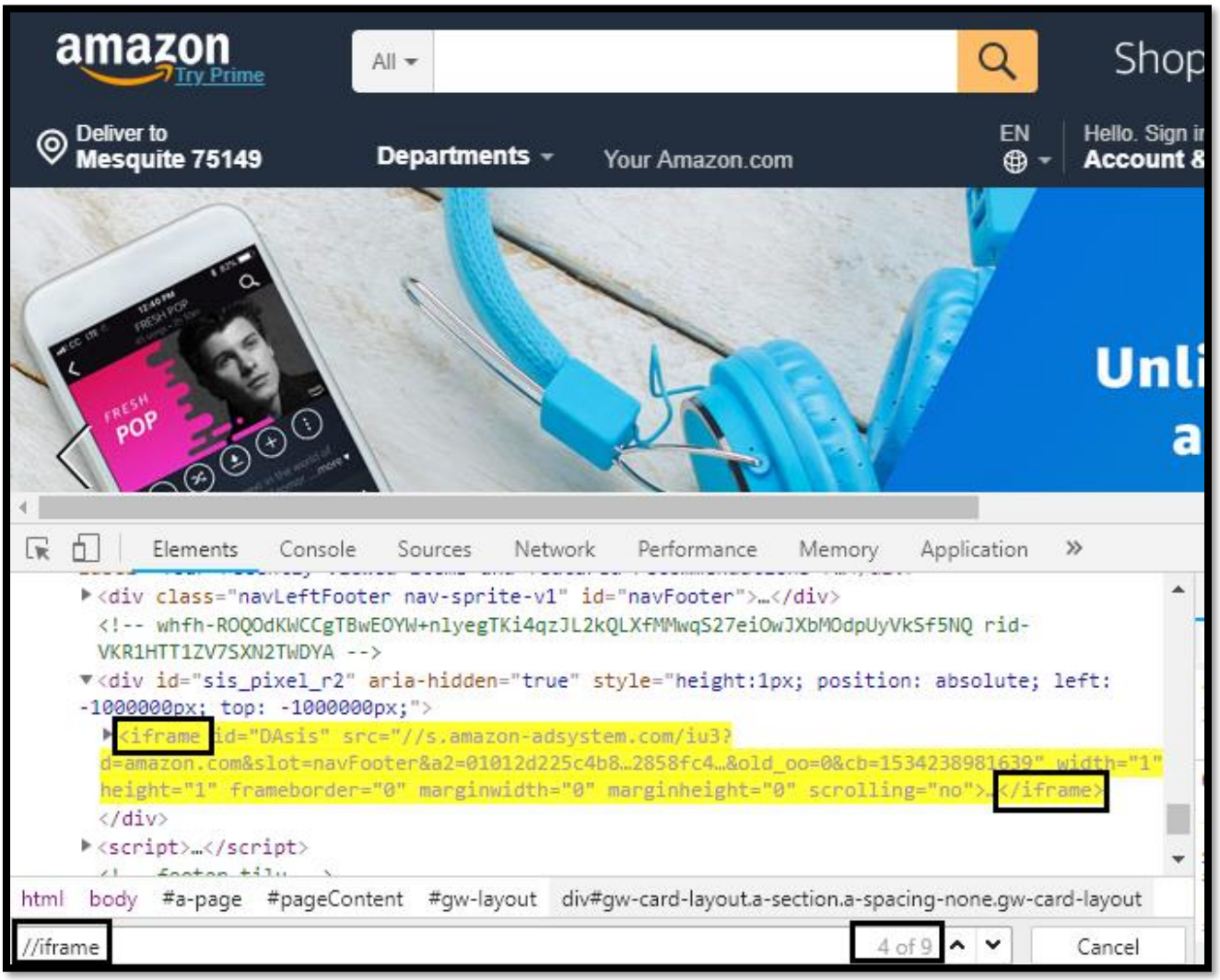


Figure 15.7 – iframe Example via Amazon

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

The diagram shows a total of 9 iframes. That particular iframe 4 of 9 is an advertisement. The following is 1 way to switch back to the first page after switching from a frame and 3 ways for switching to a frame using switchTo().frame():

1. Default Content
2. Index
3. Name or ID
4. WebElement

Index

Index switches a frame using a zero (0) based integer. The first screenshot displays syntax and description for switchTo().frame(int index) followed by switchTo().defaultContent(). Afterwards, the next 3 screenshots show part of a web page, code example, and results from the code example.

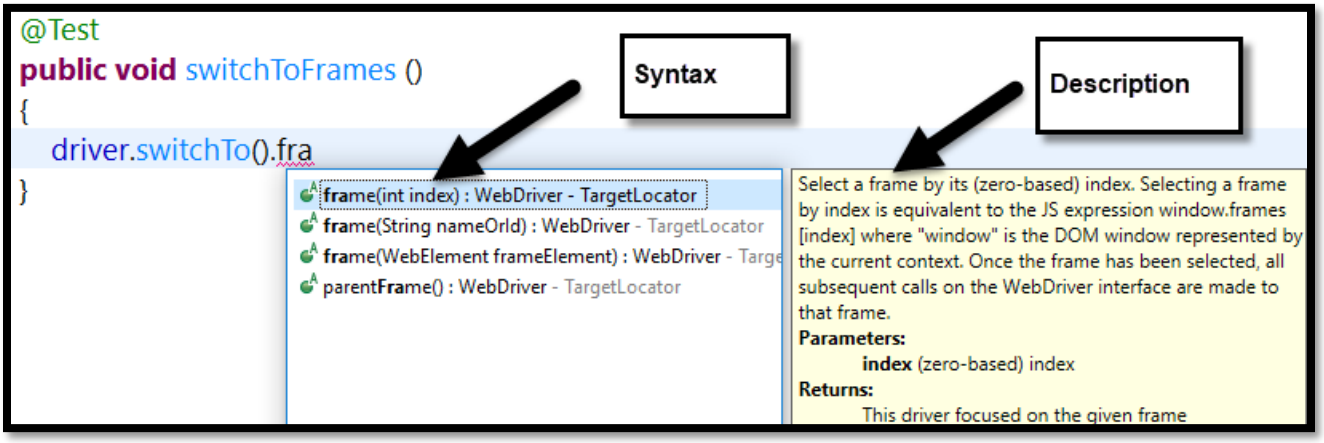


Figure 15.8 – Frame Index Syntax and Description

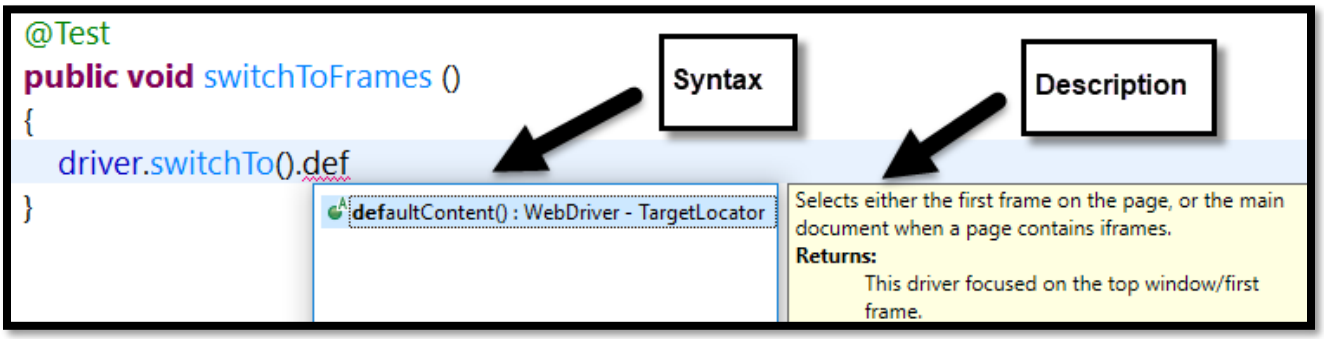


Figure 15.9 – Default Content Syntax and Description

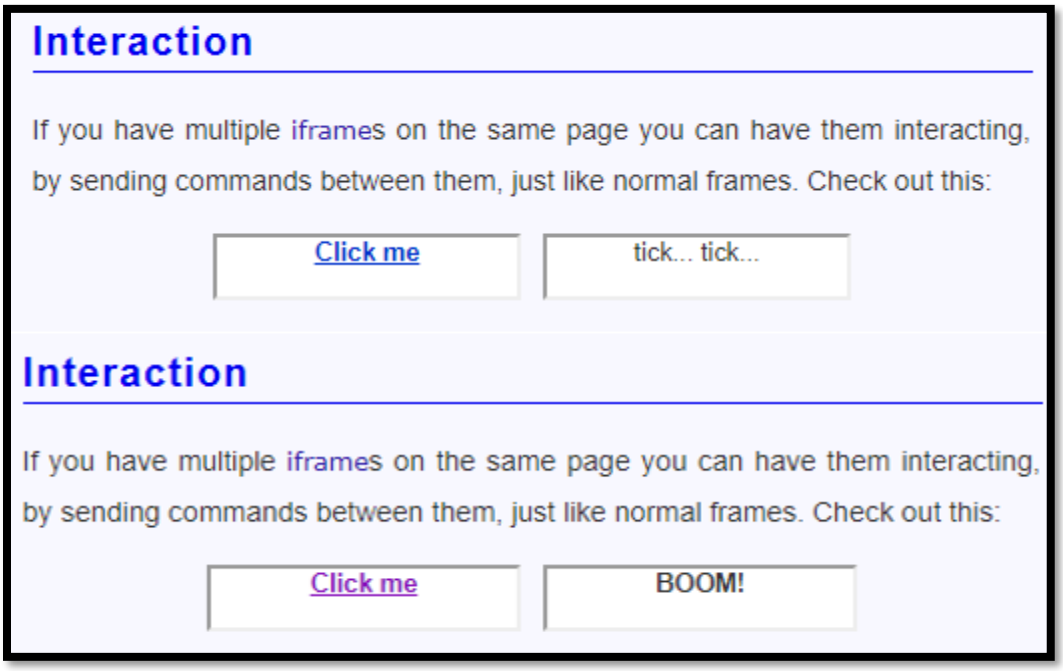


Figure 15.10 – iframes Before & After Clicking 'Click Me' via HTML Source Web Page

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

```
58  @Test
59  public void switchToFrames_Index ()
60  {
61     driver.get("https://www.yourhtmlsource.com/frames/inlineframes.html");
62
63     List <WebElement> iFrames = driver.findElements(By.tagName("iframe"));
64     System.out.println("How Many Frames Are Located On The Web Page? " + iFrames.size());
65
66     for (int i = 0; i < iFrames.size(); i++)
67     {
68        driver.switchTo().frame(i);
69
70        int frameIncludeClickMe = driver.findElements(By.xpath(".//*[@id='htmlsource']/a")).size();
71
72        System.out.println("How Many Times frame(" + i + ") Include The Words 'Click Me'? " + frameIncludeClickMe);
73
74        driver.switchTo().defaultContent();
75     }
76
77     driver.switchTo().defaultContent();
78
79     driver.switchTo().frame(2);
80
81     driver.findElement(By.xpath(".//*[@id='htmlsource']/a")).click();
82  }
```

Figure 15.11 – Example Code For Switching To A Frame Using Index

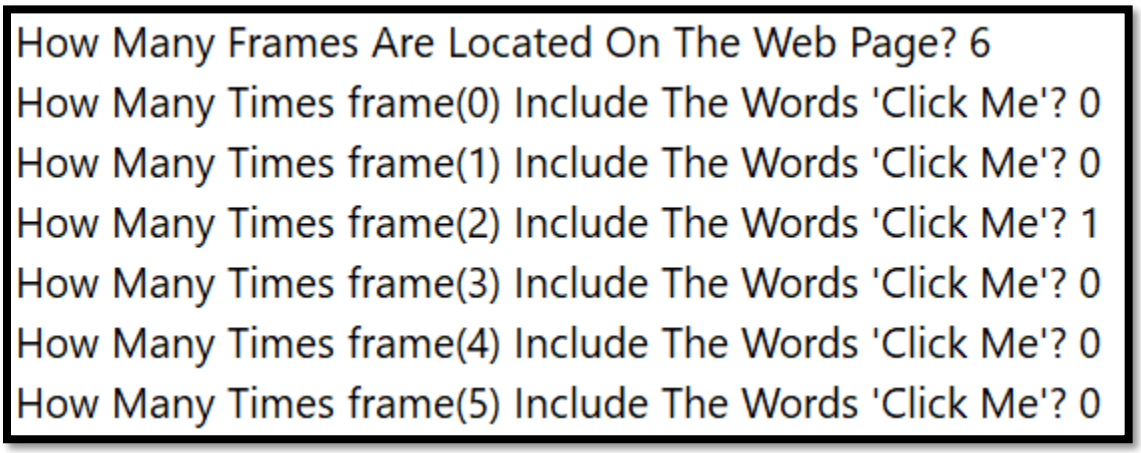


Figure 15.12 – Results Show "frame(2)" Has The Words Click Me

Figure 15.10 shows a 'Click Me' hyperlink within an iframe tag. The goal is to find the iframe tag then click the hyperlink "Click Me". After clicking the hyperlink, the words 'tick… tick…' changes to 'BOOM!'.

Automation code via Figure 15.11, switches to frame(2) notated by Line 79. Prior to switching to frame(2), all iframes are located by statement Line 63. Next Lines 66 – 75, cycle through all iframes via for loop in order to detect the words 'Click Me'. Notice, the results screenshot via Figure 15.12, shows there are 6 frames on the web page and 3rd element has the words 'Click Me'. Number 1 indicates the 3rd element which is frame(2) when starting by index 0 contains 'Click Me'. Eventually the code switches back to the main page via Line 77 then switches to frame(2) via Line 79. Finally, the hyperlink 'Click Me' is clicked via Line 81.

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Name or ID

Frames can be switched using its name Attribute or ID Attribute. The following screenshots display syntax and description for switching to a frame by name or id then part of a web page. Additionally, the last 2 screenshots show a name attribute for iframe and automation code for switching to a frame using name.

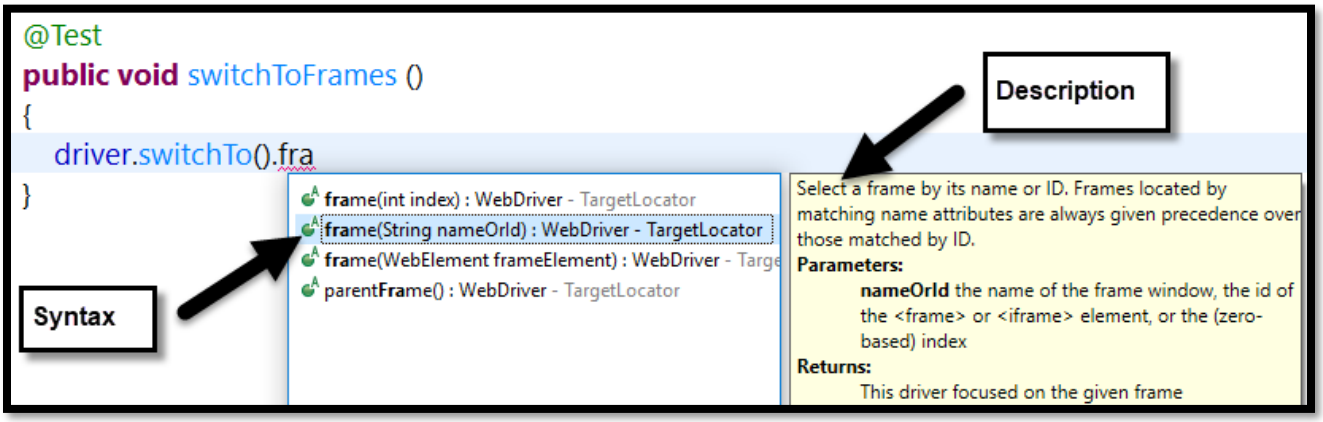


Figure 15.13 - Frame Name or ID Syntax and Description

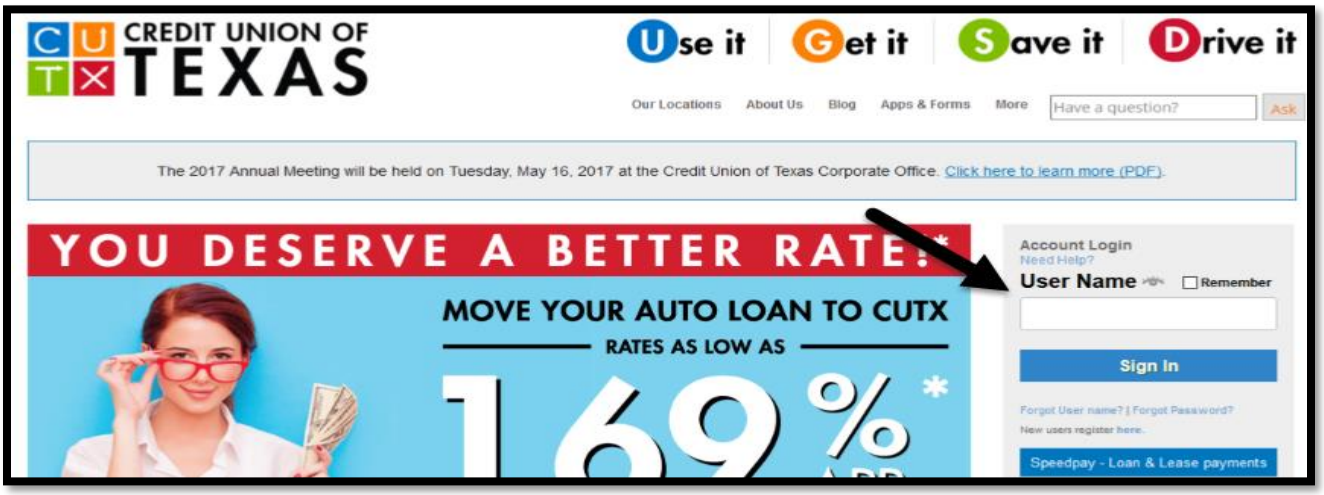


Figure 15.14 – iframe Contains User Name Text Box via Credit Union of Texas Web Page

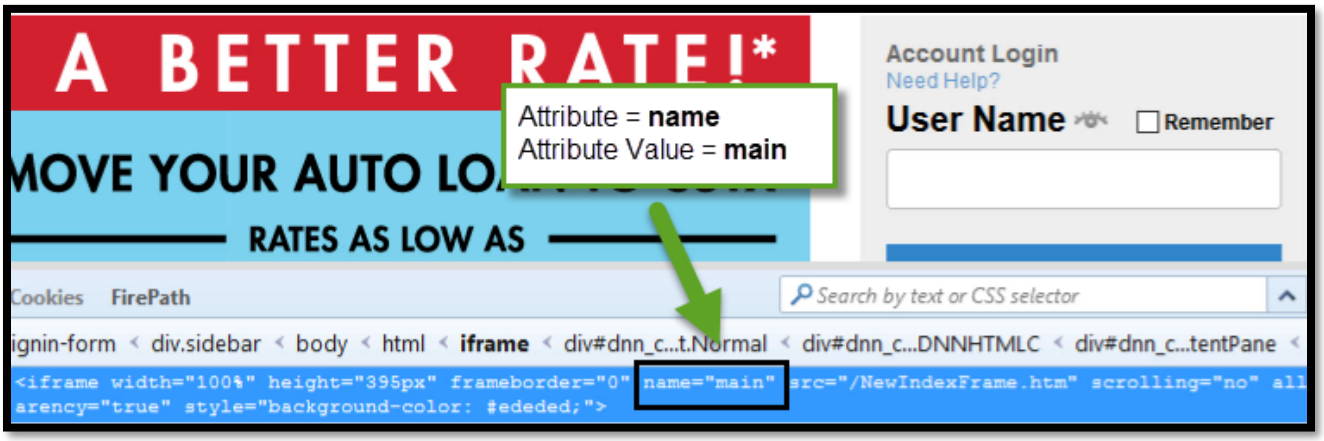


Figure 15.15 – Attribute Name Has A Value Of main

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

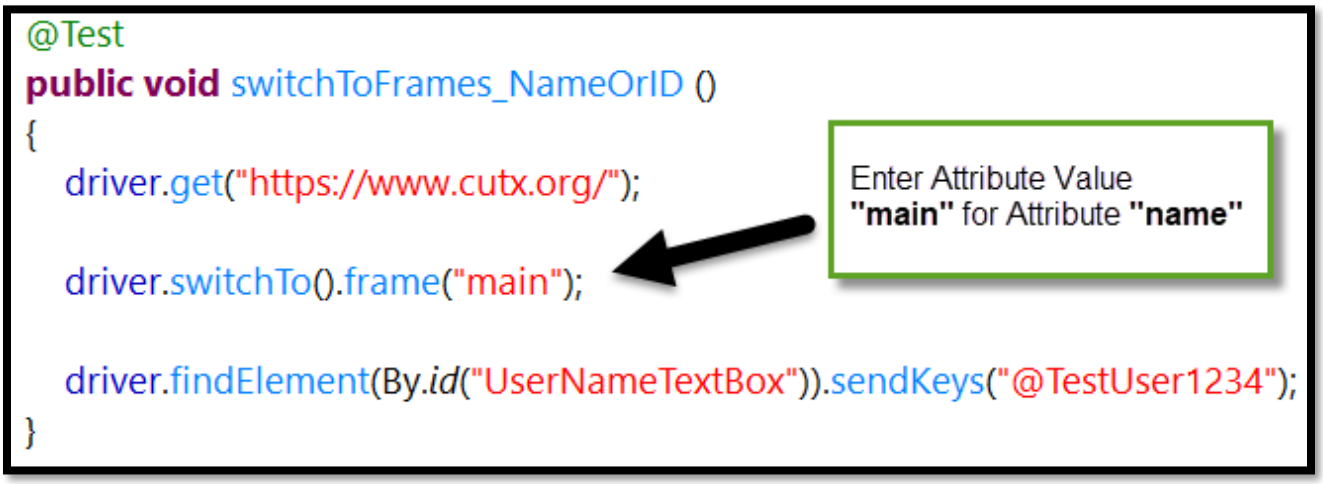


Figure 15.16 – Example Code For Switching To Frame Using Name Attribute

Figure 15.15 shows the name Attribute has a value of main. After the locating the value, it can be used to switch to a frame – switchTo().frame("main") in Figure 15.16. Lastly, the keys are sent to User Name text box. An exception shows up if the switchTo() method is not implemented before entering a value into the text box.

WebElement

Switching to a frame using WebElement has the same steps as finding an element then performing an action on the element. An automation engineer must switch to the element before performing an action.

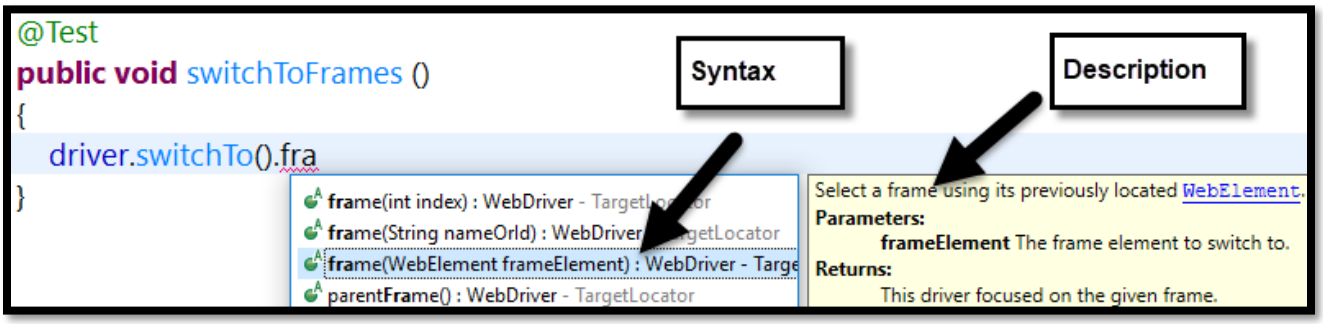


Figure 15.17 – Frame WebElement Syntax and Description

Figure 15.18 – iframe Shows Up After Clicking Log In/Sign Up via Paytm Web Page

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Figure 15.19 – iframe Is A WebElement With Additional WebElements "Text Boxes, Buttons, etc."

```
42  @Test (priority = 2)
43  public void switchToFrames_WebElement ()
44  {
45     driver.get("https://paytm.com/");
46
47     driver.findElement(By.xpath(".//*[@id='app']/div/div[2]/div[2]/div[3]/div[3]/div")).click();
48
49     WebElement login = driver.findElement(By.xpath("//iframe[contains(@src,'login')]"));
50
51     driver.switchTo().frame(login);
52
53     driver.findElement(By.xpath(".//*[@id='input_0']")).sendKeys("Joe_Doe@Email.com");
54  }
```

Figure 15.20 – Example Code For Switching To Frame Using WebElement

The iframe shows up via Figure 15.19 after clicking to log into the application via Figure 15.18. In Figure 15.20, Line 49 finds the iframe that contains login within its src Attribute. After finding the element, it is assigned to a WebElement named login. Next on Line 51, the switchTo().frame(login) statement switches to the iframe using WebElement login. Finally, an email address is entered into the text box on Line 53.

Alerts

An alert is a pop-up box that provides information or expect an action from the user. Other parts of the web page cannot be accessed until pop up box is closed by the user. To close the box, an automation engineer switches to the box then perform actions on one of the following alert types:

1. Information Alert – contains a message with 1 button "i.e., OK"
2. Confirmation Alert - contains a message with 2 buttons "i.e., OK and Cancel"
3. Prompt Alert – contains a message with 2 buttons "i.e., OK and Cancel" and text field

All alert types are created with JavaScript inside HTML. Therefore, the alerts cannot be identified with an inspector tool or a Selenium WebDriver locator type. In spite of that, the following actions either click a button, get text, or type text into the alert type:

1. accept () – accepts the alert
2. dismiss () – dismisses / cancels the alert
3. getText () – gets text from the alert
4. sendKeys () – types text into the alert

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

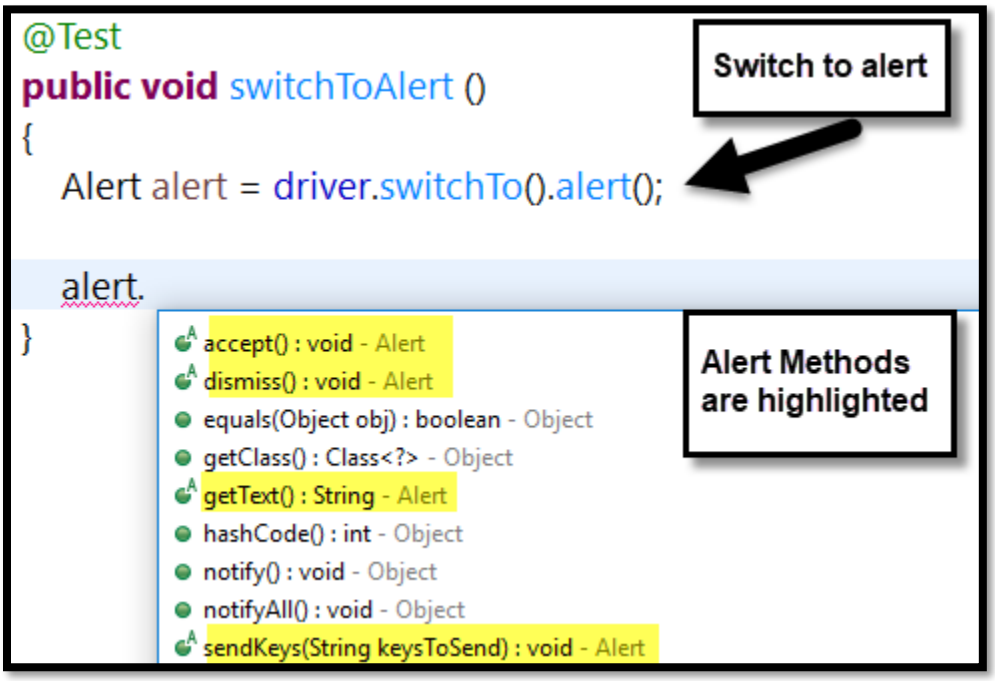


Figure 15.21 – switchTo().alert and Alert Methods

Information Alert

An Information Alert contains 1 button while providing information to the user. The following are screenshots of a sample Information Alert and automation code illustrating how to click OK button via accept() method.

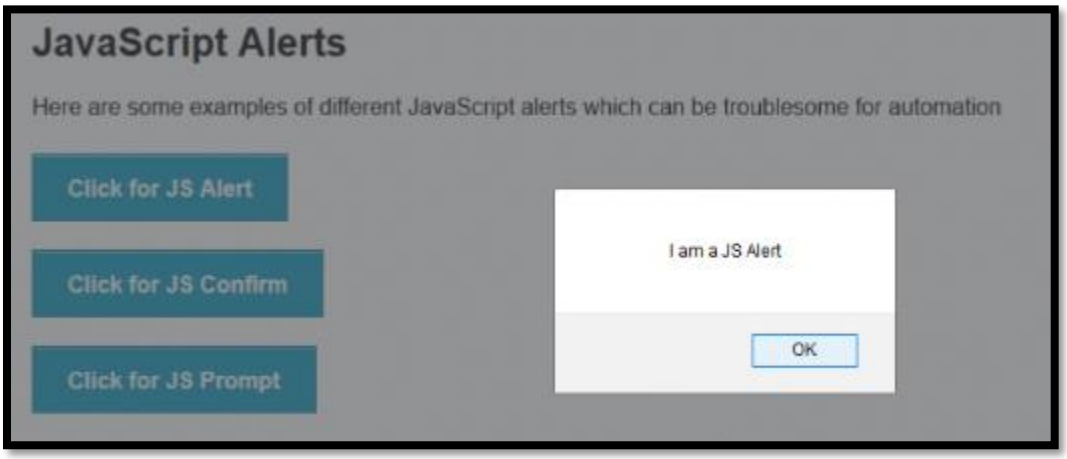


Figure 15.22 – Information Alert via Internet Herokuapp Web Page

```
37   @Test
38   public void switchToAlertDialog () throws InterruptedException
39   {
40      driver.findElement(By.linkText("JavaScript Alerts")).click();
41
42      driver.findElement(By.xpath(".//*[@id='content']/div/ul/li[1]/button")).click();
43
44      WebDriverWait wait = new WebDriverWait (driver, 5);
45
46      wait.until(ExpectedConditions.alertIsPresent());
47
48      Alert alert = driver.switchTo().alert();
49
50      Thread.sleep(2000);
51
52      alert.accept();
```

Figure 15.23 – Example Code To Accept The Information Alert

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

In Figure 15.22, the Information Alert contains a message that states “I am a JS Alert”. JS stands for JavaScript. Notice how focus is taken away from the main page which is gray while the alert is white. In order to proceed, a user must click the OK button. Figure 15.23 instantiates Alert alert and switch focus to the alert type on Line 48. Afterwards, execution pauses for 2 seconds at Line 50 when using Thread.sleep(2000). Finally, the OK button is clicked at Line 52 via alert.accept().

Note: Thread.sleep(2000) is a Wait Method that’s rarely used in automation code. It receives a parameter to sleep in milliseconds “2000 milliseconds = 2 seconds”.

Confirmation Alert

A Confirmation Alert contains 2 buttons while providing information to the user. The following are screenshots of a sample Confirmation Alert and automation code illustrating how to click Cancel button via dismiss() method.

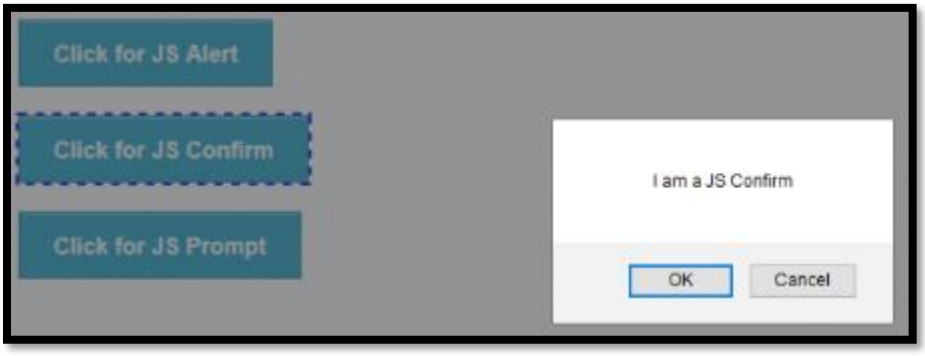


Figure 15.24 – Confirmation Alert via Internet Herokuapp Web Page

```
63  @Test
64  public void switchToConfirmDialog () throws Exception
65  {
66      driver.findElement(By.partialLinkText("Alerts")).click();
67
68      driver.findElement(By.xpath(".//*[@id='content']/div/ul/li[2]/button")).click();
69
70      WebDriverWait wait = new WebDriverWait(driver, 5);
71      wait.until(ExpectedConditions.alertIsPresent());
72
73      Alert alert = driver.switchTo().alert();
74
75      Thread.sleep(2000);
76
77      alert.dismiss();
```

Figure 15.25 – Example Code To Dismiss The Confirmation Alert (1)

```
63  @Test
64  public void switchToConfirmDialog () throws Exception
65  {
66      driver.findElement(By.partialLinkText("Alerts")).click();
67
68      driver.findElement(By.xpath(".//*[@id='content']/div/ul/li[2]/button")).click();
69
70      WebDriverWait wait = new WebDriverWait(driver, 5);
71      wait.until(ExpectedConditions.alertIsPresent());
72
73      Thread.sleep(2000);
74
75      driver.switchTo().alert().dismiss();
```

Figure 15.26 – Example Code To Dismiss The Confirmation Alert (2)

In Figure 15.24, the Confirmation Alert contains a message that states “I am a JS Confirm”. The alert also has an OK and Cancel button. Figure 15.25 and Figure 15.26 shows how to click the Cancel button using dismiss() method. However, the OK button can be clicked using the accept() method just like with Information Alert. The difference between both figures is switchTo().alert() and dismiss() methods.

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

- Figure 15.25 separates the switchTo().alert() and dismiss() methods. Line 73 displays the switchTo().alert() method while Line 77 displays dismiss() method
- Figure 15.26 combines the switchTo().alert() and dismiss() methods. Line 75 displays switchTo().alert().dismiss()

Note: Both ways are valid for dismissing an alert.

Prompt Alert

A Prompt Alert contains 2 buttons while providing a text box and information to the user. The following are screenshots of a sample Prompt Alert and automation code illustrating how to enter text via sendKeys() method.

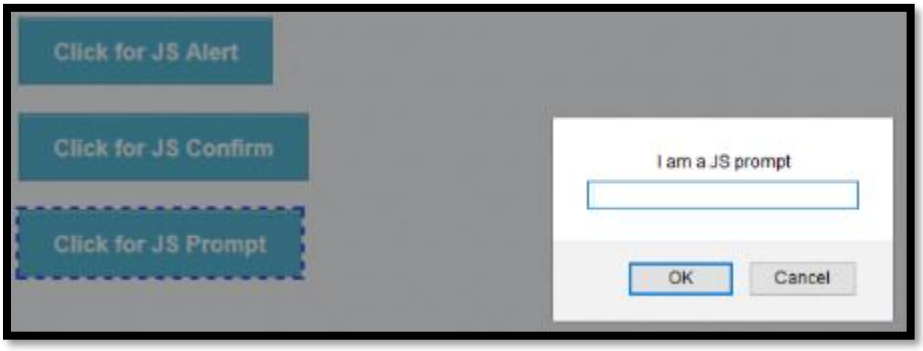


Figure 15.27 – Prompt Alert via Internet Herokuapp Web Page

```
97    @Test
98    public void switchToPromptAlert() throws InterruptedException, Exception
99    {
100       driver.findElement(By.partialLinkText("Alert")).click();
101
102       driver.findElement(By.xpath("//*[@id='content']/div/ul/li[3]/button")).click();
103
104       WebDriverWait wait = new WebDriverWait(driver, 5);
105       wait.until(ExpectedConditions.alertIsPresent());
106
107       Alert alert = driver.switchTo().alert();
108
109       Thread.sleep(2000);
110
111       String message = alert.getText();
112       System.out.println(message);
113
114       alert.sendKeys("I Am An Automation Tester");
115
116       Thread.sleep(2000);
117
118       alert.accept();
119   }
```

Figure 15.28 - Example Code To Get Text And Send Keys

```
I am a JS prompt
PASSED: switchToPromptAlert
```

Figure 15.29 – Results After Getting Text And Sending Keys via Prompt Alert

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

In Figure 15.27, the Prompt Alert contains a message that states “I am a JS prompt”. The alert also has an OK and Cancel button along with a text box. Figure 15.28 is the automation code for getting text and entering text into the Prompt Alert. Line 107 switches to the alert while Line 111 gets text from the alert via getText() method. The alert message is printed via Line 112 and displayed in Figure 15.29. Afterwards, Line 114 enters text via sendKeys() method then accepts the alert at Line 118.

Conclusion

Selenium WebDriver is a very popular open source automation effort in the Software QA/Testing industry. The goal of "*(Part 1) Selenium WebDriver for Functional Automation Testing*" was to provide a foundation for beginners. Automation engineers build on top of that foundation with knowledge of the 8 Locator Types, how to inspect a WebElement, and Selenium WebDriver's 5 Method Categories. The following are take-away topics from this book:

Selenium WebDriver is an automation effort used for testing web applications. It generates a call to a browser utilizing each browser's native support for automation. It is important to know that all web applications and browsers contain WebElements "i.e., buttons, links, etc." The WebElements are derived from HTML, which is an acronym for Hyper Text Markup Language.

HTML is the standard language for creating web applications. As a result, the key to automating an application is to first find the WebElement via HTML. After locating the WebElement, an automation engineer decides the appropriate action to perform on the WebElement. Finding the WebElement and performing an action on the WebElement are building blocks to test automation. A WebElement is found using the following 8 locator types:

1. ID - Find WebElements by the value of its ID attribute
2. Name - Find WebElements by the value of its Name attribute
3. XPath - Find WebElements by its XPath
4. CSS Selector - Find WebElements by the CSS Selector's engine
5. Link Text - Find hyperlink WebElements by its complete text
6. Partial Link Text - Find hyperlink WebElements by partial text contained within the complete text

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

7. Tag Name - Find WebElements by its tag name

8. Class Name - Find WebElements by the value of its Class attribute

The following are Selenium WebDriver's Method Categories:

1. Browser Methods – group of methods that perform an action on the browser

2. Wait Methods – group of methods that pause between execution statements

3. Navigation Methods – group of methods that load a web page, refresh a web page, move backwards in a browser's history, and move forward in a browser's history

4. WebElement Methods – group of methods that perform an action on WebElements

5. Switch Methods - group of methods used for switching to windows, frames, and alerts

Resources

1. Selenium HQ
   http://www.seleniumhq.org/
   http://www.seleniumhq.org/about/history.jsp
   http://www.seleniumhq.org/download/
   https://seleniumhq.github.io/selenium/docs/api/java/index.html?org/openqa/selenium/WebDriver.html

2. W3C
   http://www.w3.org/TR/WCAG20-TECHS/H93.html

3. LinkedIn
   https://www.linkedin.com/

4. Facebook
   https://www.facebook.com/

5. Yahoo
   https://www.yahoo.com/

6. WordPress
   https://login.wordpress.org/

7. Welcome to the Internet
   http://the-internet.herokuapp.com/

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

The following resources are links to installations in Chapter 1.

8. Install Java Development Kit (JDK) via Oracle
   http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

9. Install Eclipse IDE via Eclipse
   https://eclipse.org/downloads/

10. Install Java Client Driver via Selenium HQ
    http://docs.seleniumhq.org/download/

11. Install ChromeDriver via Selenium HQ or Google Sites
    http://docs.seleniumhq.org/download/
    https://sites.google.com/a/chromium.org/chromedriver/downloads

12. Install IEDriverServer via Selenium HQ or Selenium Server
    http://docs.seleniumhq.org/download/
    http://selenium-release.storage.googleapis.com/index.html?path=2.53/

13. Install Notepad ++
    https://notepad-plus-plus.org/download/v6.9.2.html

Download PDF Version

The PDF Version of this book is available to you at the following link:

http://tinyurl.com/Part-1-Selenium-WebDriver

If the book was helpful, can you leave a favorable review?

http://tinyurl.com/Review-Pt-1-Selenium-WebDriver

Thanks in advance,

Rex Allen Jones II

Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Books by Rex Jones II

www.tinyurl.com/Rex-Allen-Jones-books

1. **Free Book** Absolute Beginner
   (Part 1) You Must Learn VBScript for QTP/UFT
   Don't Ignore The Language For Functional Automation Testing

2. (Part 2) You Must Learn VBScript for QTP/UFT
   Don't Ignore The Language For Functional Automation Testing

3. **Free Book** Absolute Beginner
   (Part 1) Java 4 Selenium WebDriver
   Come Learn How To Program For Automation Testing

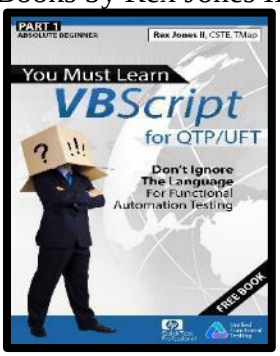
4. (Part 2) Java 4 Selenium WebDriver
   Come Learn How To Program For Automation Testing

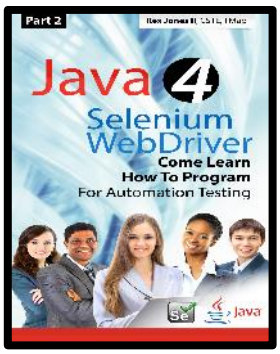
5. **Free Book** Absolute Beginner
   (Part 1) Selenium WebDriver for Functional Automation Testing
   Your Beginners Guide

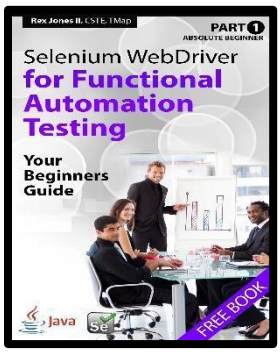
6. Getting Started With TestNG
   A Java Test Framework

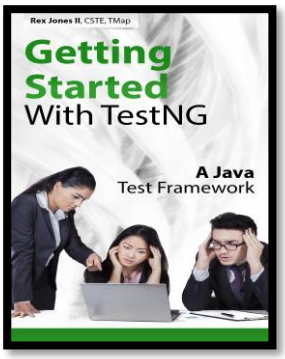
3 Tips To Master Selenium Within 30 Days
http://tinyurl.com/3-Tips-For-Selenium

Free Webinars, Videos, and Live Trainings
http://tinyurl.com/Free-QTP-UFT-Selenium









Skype: rex.jones34
Twitter: @RexJonesII
Email: Rex.Jones@Test4Success.org
LinkedIn: https://www.linkedin.com/in/rexjones34

Sign Up To Receive

1. 3 Tips To Master Selenium Within 30 Days
   http://tinyurl.com/3-Tips-For-Selenium

2. 3 Tips To Master QTP/UFT Within 30 Days
   http://tinyurl.com/3-Tips-For-QTP-UFT

3. Free Webinars, Videos, and Live Trainings
   http://tinyurl.com/Free-QTP-UFT-Selenium

3 Tips To Master Selenium Within 30 Days
http://tinyurl.com/3-Tips-For-Selenium

Free Webinars, Videos, and Live Trainings
http://tinyurl.com/Free-QTP-UFT-Selenium