Bachelor's thesis

Information Technology

Software Engineering

2014

Md.Foyzur Rahman

BROWSER-BASED AUTOMATION TESTING USING SELENIUM WEBDRIVER

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR´S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES

Degree programme | Information Technology

June 2014| 85

Instructor: Ferm Tiina

Md.Foyzur Rahman

BROWSER-BASED AUTOMATION TESTING USING SELENIUM WEBDRIVER

This thesis deals with automation testing for any web applications on different web browsers using Selenium WebDriver. Automating web applications has been a very important and challenging issue now-a-days for QA engineers and software developers testing web applications. This thesis also briefly describes the advantages and disadvantages of automation and manual testing and in which test cases they can be applied to achieve high quality software. This thesis focuses on automating Gmail on the Firefox and Chrome browsers.

Considering the requirements for test automation, Selenium WebDriver is an open source tool that can mimic a real end user and was chosen for its flexibility, language bindings, functionality, and growing community .WebDriver has the capability of driving multiple browsers or different versions of the same browsers on different platforms.

Test scripts for four test cases were implemented and run only on a Windows 7 platform. All test cases were executed successfully and the outcomes were identical to the predicted outcomes.

KEYWORDS:

Gmail, WebDriver, Automation Testing, Open Source, Firefox, Chrome, QA, Manual Testing

ACKNOWLEDGEMENT

First of all, I would like to thank and express my gratitude to Mrs. Ferm Tiina for making free time out of her tight schedule as well as guiding and advising me to all the way to accomplish this thesis work .I also thank all the teachers who taught and advised me during my studies at TUAS.

I also would like to specially thank Mr. Ossi Väänänen and Mr. Vesa Slotte for their help and unlimited support at TUAS Cisco Lab during my study period.

I also appreciate to all my friends as well as my family members to motivate me to complete my degree.

Last not the least, sincerely I thank and appreciate Mr. Abdur Rahman, my maternal uncle, whose support, inspiration, and guidance has been a great influence in my life.

Rahman Md. Foyzur

Turku,Finland

June 2014.

TABLE OF CONTENTS

APPENDICES

PICTURES

FIGURES

TABLES

LIST OF ABBREVIATIONS (OR) SYMBOLS

PHP	Hypertext Preprocessor
C++	Pronounced C Plus Plus
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
OS	Operating System
RC	Remote Control
IE	Internet Explorer
AUT	Application Under Test
CLI	Command Line Interface
N/A	Not Applicable
GTAC	Google Test Automation Conference
DOM	Document Object Model
CSS	Cascading Style Sheet
TestNG	Test Next Generation
XML	Extensible Markup Language
HTML	Hyper Text Markup Language
XHTML	Extensible Hyper Text Markup Language
OS X	Known as Mac OS X
JVM	Java Virtual Machine
JDK	Java Development Kit
JSON	JavaScript Object Notation
XPCOM	Cross Platform Component Object Model
CDS	Chrome Driver Server
QA Engineer	Quality Assurance Engineer

1 INTRODUCTION

Any software, no matter what language it is implemented in, has some bugs. Some of them are detected and removed while coding. The rest are found and fixed while integrating software modules into a system during formal testing. However, it is known to all manufacturers that bugs that remained in the software can be fixed later. At this point, testing is required for a successful implementation of the software. Based on the project requirements, testing will fix all the bugs before releasing the software. However, for a most complicated project test automation is needed.

Now-a-days, there are many commercial software testing tools. A powerful and flexible software-testing tool with more features to test a large scale project should be chosen so that it can deal with browsers' rich content API along with dynamic web applications. On the other hand, a tester can focus on fixing bugs in high risk areas with such a tool in a complicated web application.

WebDriver is a new technology with a suite of various tools to automate browser-based web application. The purpose of the thesis was to automate Gmail on multiple browsers and Selenium WebDriver was chosen. The web browser can be driven by Selenium WebDriver in order to execute different types of automation tests on the web application as if an end user can navigate through them. Selenium WebDriver can emulate actions like clicking on the image, context or entering text, dragging and dropping a WebElement anywhere in the DOM, submitting forms, opening and closing a new tab or window, downloading a file from the internet and saving it to a particular location in a local machine, and reporting the results back to the end user so that end user knows that it works as expected.

Using this tool, four test scripts were written to perform functional testing on Gmail and results were documented as well. All test cases were executed on Chrome and Firefox browsers .All test cases were tested successfully. The analyzed result of the total execution time taken by Chrome and Firefox browsers is given in this thesis.

In practice, it is unrealistic to automate everything in a web application like Gmail by WebDriver.

1.1 Thesis overview

The purpose of this thesis is to automate a certain portion of Google Mail (Gmail) using WebDriver. The thesis is structured as follows:

Chapter 1, Introduction, gives a clear overview of the thesis. This chapter explains the goal for the project as well as the motivation for the project.

Chapter 2, Automation Testing, introduces test automation including the advantages and the drawbacks as well as in which web application test automation can be executed .

Chapter 3, Selenium WebDriver, starts off by discussing briefly about Selenium, its history, architectural overview, Selenium RC , the architectural structure of Selenium RC ,Selenium Server , Selenium WebDriver , and the platforms supported by WebDriver. Then, readers can also have a clear conception about WebElements and how to locate an element in the DOM and then perform some actions on them.

Chapter 4, WebDriver and its features, discusses the types of testing depending on the web applications. It dives deeply into the advance interactions of WebDriver that can be performed on the WebElements in the DOM. It also briefly explores some of the advance features of WebDriver as well as different types of WebDrivers and how they work.

Chapter 5, Testing and Reporting, deals with the test design, implementation, and running the test cases in Firefox and Chrome browsers. The test cases results are presented as a console-based print out and HTML-based reports. There are some common problems and their solution is discussed.

Chapter 6, Conclusion, summarizes the goal of the project and a possible future work proposal.

2 AUTOMATION TESTING

Test automation is a process in which a testing tool is used to compile and execute a test on a software application. A test suite can be carried out repeatedly with these tools. Automation testing tools execute the complicated and large scale tests, generate the reports, and finally compare with the test results run earlier. The target of automation testing is to focus on problems mostly in a certain part of the software (Li, 2004, p10). A test script can have a few repetitive code patterns in order to test a variety of methods inside a class. In automation testing, no manual intervention is needed when an automated test is running a test suite .Test automation is more reliable, programmable , reusable, comprehensive , maintainable , saves money and time , decreases cost, has greater test coverage and is faster than human interactions - compared to manual testing. A tester writes the test cases and then splits them into many test suites because a smaller test suite runs faster than a large test suite and is easy to maintain the test script. In Software Organizations, test automation has been very important in software development. (Li, 2004, p5)

The following diagram, demonstrates the steps to automate software during the development period.

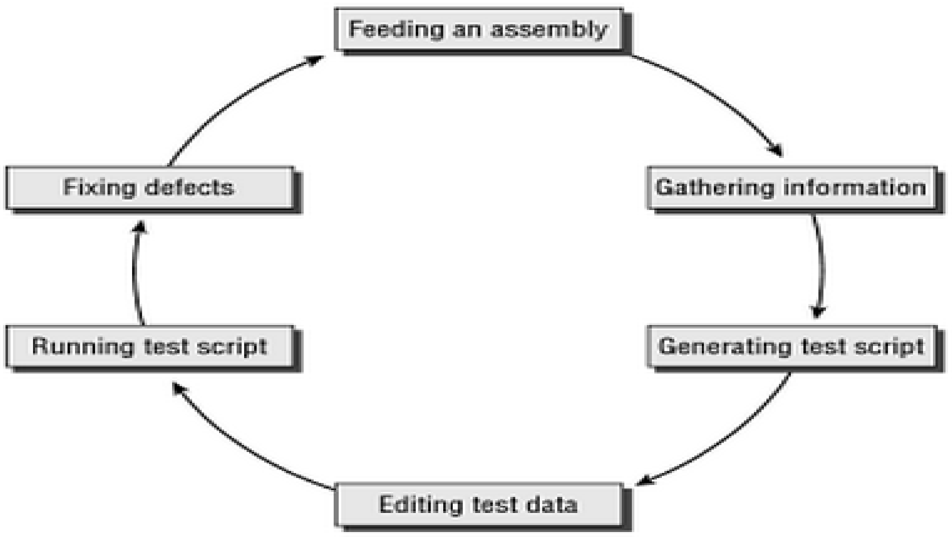


Figure 2.0 Steps for the automated software testing (Li, 2004, p9)

2.1 When to automate

Automation testing can be done if the test is run frequently in the same application or on a certain portion of the application. It is almost impossible to automate all kinds of testing, for example, usability testing; however, if the workload and costs are too high for manual testing; automation testing can be taken into consideration. For instance, sanity testing, regression testing, data driven testing and batch testing can be automated. Automation testing can be carried out on any application:

- Large and complicated projects
- Stable software/application
- Frequently testing same test cases
- Software runs on cross platform
- Time consuming
- High risk of having error in manual testing

2.2 When not to automate

Test automation cannot be done in the following cases:

- If the test requires observation from end user.
- If the test is run only once.
- On any unstable software/application.
- By any inexperienced or temporary tester.
- If not having enough time to build automation testing. (Introduction-Selenium Documentation, 2014)

2.3 Drawbacks of automation testing

Automation testing has its own drawbacks too even though it has many advantages. These disadvantages include:

- Highly skilled Software tester required.
- Testing tool itself can have bugs.
- Major issue is debugging
- Test maintenance is costly ,for example, playback methods

3 SELENIUM

Selenium is a software testing framework for web applications that provides unified interface and works with almost all browsers and tests can be compiled in many languages. Selenium is composed of several components such as Selenium IDE, Selenium WebDriver , Selenium Client API, Selenium Grid and Selenium RC(Selenium RC is deprecated recently in favor of Selenium WebDriver). Selenium IDE is a Firefox extension that allows a tester to record, edit, and debug tests . Selenium core is included in the Selenium IDE that allows to record, play back, and save a test easily and quickly in the real environment while running. Selenium is open source software released under the Apache 2.0 License. This software runs on different platforms such as Linux/UNIX, Windows and OS X platforms. It is considered as one of the first popular Open Source Projects which facilitated eventually browser-based testing for the software testers and developers and any newly released browsers can be added for support quickly and easily because it is written in pure JavaScript.

Since the Selenium engine written in JavaScript, it causes a significant weakness because very strict security models are imposed by the browsers on any JavaScript that are executed by the browsers to protect users from malicious scripts. For instance, in Internet Explorer (IE) it is harder to upload a file because IE prevents JavaScript from changing any value of any element and navigating between domains because of IE's its single host origin policy problem.

The main task of Selenium Grid is to run tests side-by-side on different machines against different browsers or different versions of browsers and operating systems using WebDriver. Selenium Grid allows a tester to perform distributed test execution by which time can be reduced to complete tests suite.

3.1 Selenium WebDriver

WebDriver, also commonly known as"Selenium WebDriver" or sometimes "Selenium 2", (Avasarala, 2014.p12) is a browser test automation tool for web applications that is used for compiling end-to-end tests. WebDriver, which is designed in a very simpler way to provide a user friendly API to make it easier to use than the Selenium RC 1.0 API, is more concise programming interfaces. This tool does exactly what an end user would expect using a browser: the control of a browser is automated so that an end user would iterate the automated tasks. This seems like such a simple problem to resolve but behind the scenes, several tasks have to be carried out before making it work. WebDriver is the name of the interface. It drives the browser either by a native user or by a remote user much more effectively. Thus, the limitations of Selenium 1.0, that affected the functional test coverage such as file uploads, downloads, pop ups and dialogue barrier, are overcome. Selenium WebDriver has incorporated the language bindings and the implementations of the individual browser controlling code. WebDriver is just one component of selenium among many. WebDriver does not need to connect to a Remote Server to perform its task in the native machine unlike Selenium RC.

Selenium WebDriver is completely an object-oriented API if we compare to Selenium 1.0. Let us have a look below:

Selenium 1.0 + WebDriver = Selenium 2.0 (Selenium WebDriver, 2014. Selenium WebDriver)

In the summer of 2011, Selenium 2 was released. The implementations of WebDriver normally differ in each browser and drive the browser in order to test in a native machine with a specific browser and imitate human interactions with the web applications.

From WebDriver architecture, it is known that WebDriver directly calls the browser when tests are performed in a native machine by using the browsers built-in JavaScript support for the automation unlike Selenium RC. A point to be noted is that in Selenium RC, the architecture works in a different way where it passes the HTTP requests to the server and the server passes it to the Selenium Core. Conversely, WebDriver does not have any proxy server in between the client and the browser. It directly passes the Selenese commands to the Selenium Core. So, The WebDriver API is a more powerful tool to speed up the execution time of tests compared to Selenium RC. Mobile browser-based tests were not possible at all before by using Selenium RC but WebDriver can interact with the browser’s rich content API and mobile applications. WebDriver supports almost all the latest versions of browsers out there in the market. It is officially stated that all future enhancements can be done only in WebDriver.

3.2 Architecture of WebDriver

We already know that Selenium RC has been replaced by WebDriver; the architecture of web driver differs from Selenium RC. Selenium RC was built in JavaScript to emulate user actions. Literally, the JavaScript drives the browser automatically from within the browser. Conversely, in the case of web driver, it drives the browser from outside the browser by using accessibility API.

Firefox uses JavaScript to access the API but IE uses C++.However, WebDriver uses the most proper way to access the accessibility API. So, we have known that this approach will not work if this applies to a new browser especially those entered the market right away because these browsers have rich content API.

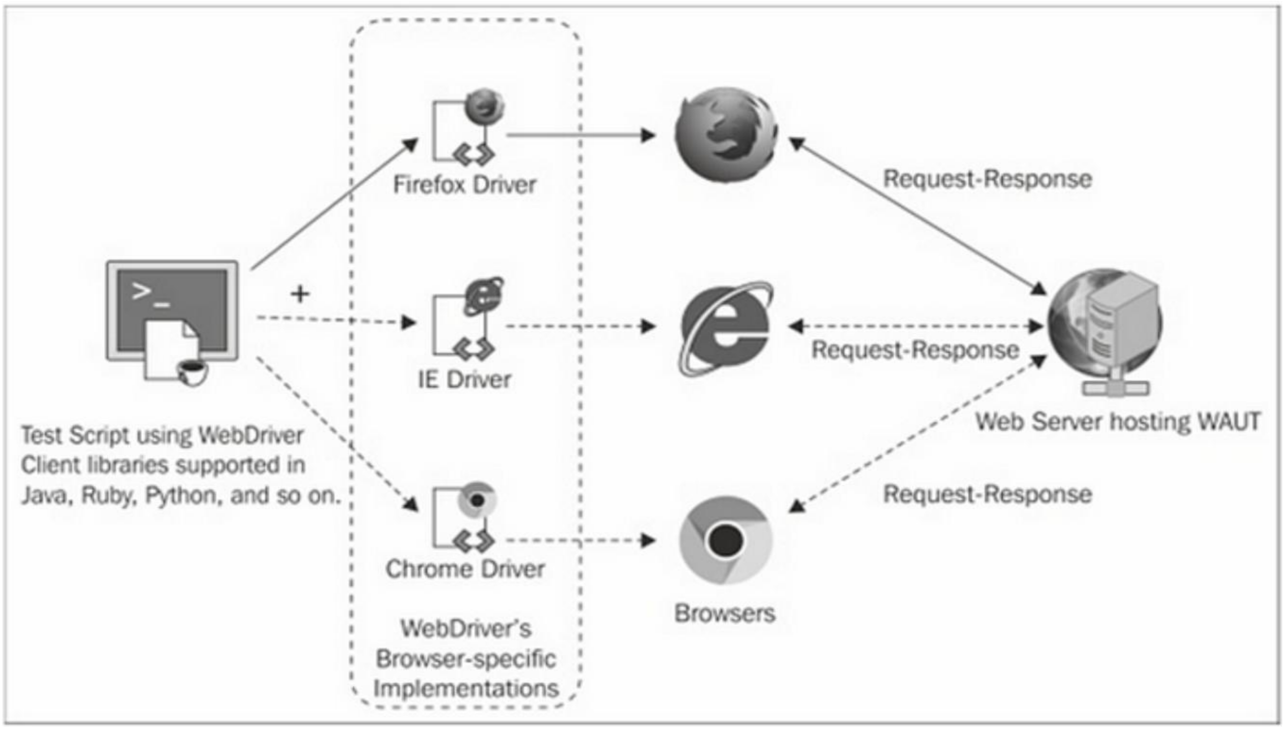

Firefox Driver
IE Driver
Chrome Driver
WebDriver's Browser-specific Implementations
Test Script using WebDriver Client libraries supported in Java, Ruby, Python, and so on.
+
Request-Response
Request-Response
Request-Response
Web Server hosting WAUT
Browsers

Figure 3.2. WebDriver Architecture (Avasarala, 2014.p-13)

Selenium Server might be needed or might not be needed depending on the test cases meaning how a tester intends to use the WebDriver. We definitely do not need the server if the WebDriver API is only used for the tests and the browser and tests are run on the same machine; WebDriver calls directly the specific browsers.

- The following reasons are stated to clarify where Selenium Server would be used with Selenium WebDriver: However, if the test is run on a remote machine where a client is on the native machine and the particular browser version is on the remote machine, in this scenario it is a must to use

Selenium Server with WebDriver every time server is required to start before running any test.

- If a tester does not use the Java bindings, for instance, C#, PHP, Python, and then the tester intends to use HtmlUnitDriver.
- If a tester runs the tests on multi-machines or virtual machines concurrently.

3.3 Selenium Server

Java was implemented to write Selenium Server (Richardson, 2012, p43). The set of Selenium commands used to test web application is known as Selenese. Selenese commands are received by the Selenium Server using HTTP GET/POST requests from the clients. The programs can be written in any programming languages (i.e., PHP, Perl, Python, and Java) that can send HTTP requests in order to perform tests on the browser. Selenium Server interprets them and sends reports back to the tester of the tests programme of those performed tests. Selenium Server basically bundles a set of JavaScript functions to inject it automatically into the browser while the program loads or opens up the browser. A client library API function is used to open up a browser when the programme runs on the machine. In other words, this injection only happens when the browser is loading or opening a web application to run a programme. Selenium Core is composed of a set of JavaScript functions and its main task is to interpret and execute the Selenese Commands by using the browser's built-in JavaScript interpreter.

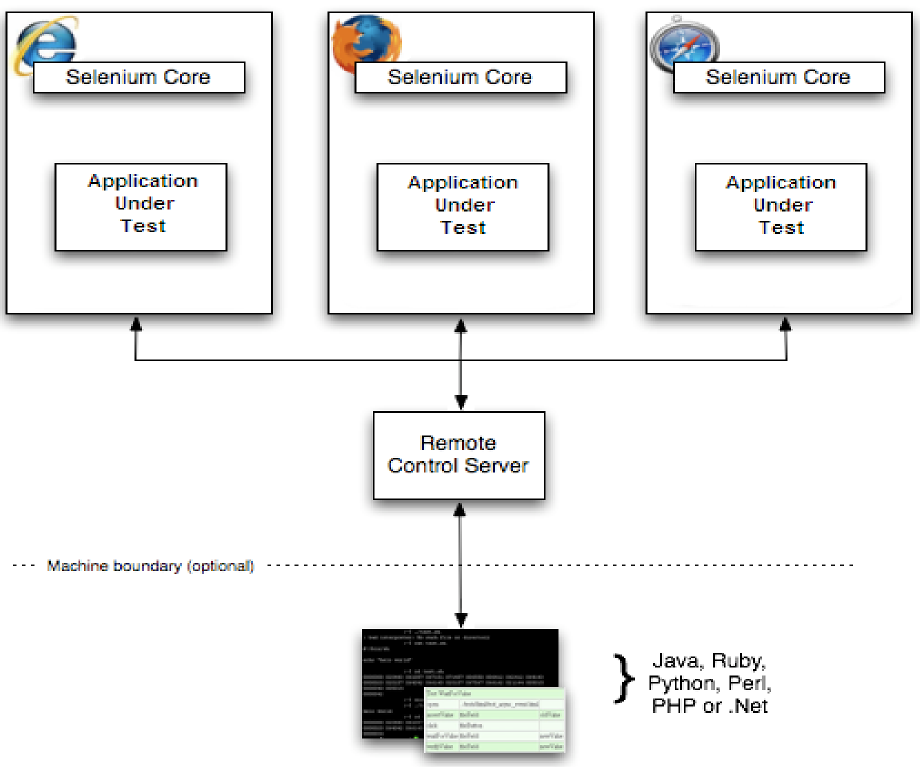


Figure 3.3. Architecture of Selenium RC (Selenium 1 (Selenium RC))

This diagram illustrates how client library passes the request to the remote server and the remote server passes each selenium commands to the browser using Selenium core. The browser uses its own interpreter to execute the Selenese commands to run the test programme.

3.4 Client Library

The programming language a tester uses to write a test is provided by the client library that allows a tester to run Selenese from a tester own design. The programme can be written in any of the following languages: Java, PHP, C#, Perl, Ruby, and Python. For each of the supported programming language, there are different client libraries. An API, for example, a Selenium Core is provided by a client library to run the Selenese commands from a written programme.

Selenese commands are taken by the client library and are sent to the server to execute test against the AUT. The results are received by the client library and are sent it back to the programme. However, the results are received by the programme. Furthermore, the results are stored into a programme. The results are reported by the programme as a success or fail. If there are some unexpected results to be found, some actions are taken possibly by the programme to correct it.

3.5 Supported Platforms by WebDriver

Selenium WebDriver, Remote Control Server and Selenium Core are compatible with many browsers, OS, programming languages and testing frameworks. The following table lists which browsers, OS, Programming languages and Frameworks are supported by WebDriver.

Supported OS By WebDriver

Table 3.5.1 Supported browsers (Platforms Supported by Selenium. 2014.)

Browsers(32 or 64 bit where applicable)	**Selenium Core**	**WebDriver API/Selenium 2**
Firefox Latest or previous version	Run Tests	Run Tests
IE 6-10 but not in IE 11(in IE 11 bug is currently found and to be fixed soon)	Run Tests	Run Tests
Google Chrome latest versions	Run Tests	Run Tests
HtmlUnit	Run Tests	Run Tests
Opera 12.x and older versions	Run Tests	Run Tests
Safari 5.1 runs only on Apple OS X	Run Tests	Run Tests
PhantomJS	Run Tests	Run Tests

Supported OS By WebDriver

Table 3.5.2 Supported OS (Platforms Supported by Selenium. 2014.)

OS	**Run Tests**
Windows 7,8,8.1	Yes
Apple OS X	Yes
Solaris	Yes
Linux	Yes
Android	Yes
iOS	Yes

Programming Languages and Frameworks Supported By WebDriver

Table 3.5.3 Supported Languages and Frameworks (Platforms Supported by Selenium. 2014.)

Language	**Frameworks**
C#	NUnit
JAVA	TestNG, JUnit
PERL	Not Supported
PYTHON	Unittest, Pyunit, Robot
RUBY	RSpec, Test::Unit
PHP	Not Supported
OBJECTIVE-C	Not Supported
HASKELL	Not Supported
JavaScript	Not Supported
R	Not Supported

3.6 Selenium WebDriver and JAVA client driver configuration

The tools are required for this project is in the table 3.6.Table 3.6 Software used in the project.

Name of the software	Version	Descriptiton
Eclipse-jee-indigo-SR2	Indigo Service Release 2	Integrated development environment for unit testing
Selenium-Java	2.40.0	A set of tools to automate the web applications
Java JDK	1.7.0_45	Set of software and tools to compile and run java programme.
Firebug	1.12.8	To inspect an element in the DOM
Firefox Browser	29.0.1 and 28.0	A popular and modern web browser
Google Chrome Browser	33.0.1750.154 m	Freeware web browser that makes web safer and faster.
FirePath	0.9.7	A Firefox extension to find out the XPath and CSS values
SelectorGadget	0.1	It is an open source Google Chrome extension to make CSS selector generation
ChromeDriver	2.8	Standalone Server that implements wire protocol for chromium
Windows 7 home Premium	SP1	Modern Operating System
TestNG	6.8.5	Unit testing Framework for Java

3.7 Finding WebElement

In WebDriver, a web page consists of different HTML elements that are known as WebElements such as body, form, input, a, iframe and so on (Avasarala, 2014. p20).In a simplified way it can be said that Locators are a way to tell Selenium which specific element we want to act on. To write a test using Selenium WebDriver, it is highly recommended to find out the correct elements from the web applications. Selenium WebDriver itself supports different options to identify the most appropriate elements. However, Selenium WebDriver rich API provides multi-locator techniques to identify elements by CSS selectors, XPath, Tag Name, Partial Link Text, ID, Name etc. Custom locator strategies could be also used to locate elements. Automated test succession totally depends on identifying and locating the right GUI (Graphical User Interface) elements from the AUT and then performing the tests and verifying different elements to achieve the test flow (Gundecha, 2012,p7).In many cases, Id or Name can be found where CSS selector or XPath are needed identify the right elements. Now-a-days, CSS selector is used more than XPATH because of its performance, simplicity, and readability. Going back to the browser concept, HTML codes with other resources of the application are hidden by the browser to render visual elements of that application for end users. It is recommended to analyze the page and locators in order to understand how AJAX and JavaScript functions are called and how page and elements are structured by using what properties and attributes before starting to explore locators (Gundecha, 2012, p8) .The best way to make it more effective and useable would be to identify a unique id's or name. To inspect an element in any of these major browsers (Firefox, Chrome, IE, and Safari) we just right click on the page and find ***inspect an element*** and then click on it. Conversely, inspecting an element can be seen by clicking tools from the menu bar and then click on ***developer tools*** on any of those brewers manually. In Firefox an extension can be installed named ***firebug to inspect element*** .On the other hand, in Chrome it can be seen only for cssSelector by using an extension named SelectorGadget. These days' software testers rely on more CSS Selector rather than choosing anything else such as ID, name, and tag name and so on; however, XPath is handier compared to cssSelector.

Selenium WebDriver supports different language bindings and in each of them, it displays methods named findElement () and findElements () in which findElement() returns object of a web element and otherwise it throws NoSuchElementFoundException error but findElements () returns an empty list if it does not find any matching of DOM elements (Unmesh ,2012,p18-20). The methods findElement () and findElements () take a query object which is known as By.

Find Web Elements by ID attribute

Using the id attribute to locate the Web Elements on the page is probably the most preferred and common approach. According to W3C standard, it is highly recommended to use a unique id attribute for each element by which it is very explicit and easy to identify elements in the page.

The following source code depicts that the input element having an id element with a value that is '"Email".

```
<input type="email" class="" spellcheck="false" value="" placeholder="Sähköposti" name="Email" id="Email">
```
(Gmail. 2014. *Gmail*.)

```
//write your Email

driver.findElement(By.id("Email")).sendKeys("thesis.web.driver");
```

Find Web Elements by Name attribute

There are some cases where id attribute is not specified for an element on a page though W3C recommends UI designers to use it. In addition to a dynamic web page, an id is always generated dynamically each time a page is loaded or refreshed. In this scenario, it is better to use a name attribute to identify web elements although a name attribute is not unique like id. There can be a scenario where multiple elements have the same name attributes and in such situation this strategy might not be able identify the desired element on the page because it takes the first element's name attribute on the page.

In the following input, the element does not have an id but having a name attribute which can be used to locate web element. This can be used like this to locate the input element. To identify the element using the name attribute can be done as follows:

```
<input type="password" class="" placeholder="Salasana" name="Passwd" id="Passwd">
```
(Gmail. 2014. *Gmail*.)

```
//Identify the element

driver.findElement(By.name("Passwd")).sendKeys("thesis.web.driver");
```

Find Web Elements by Class attribute

The main purpose of using class attribute in any web element is to apply CSS. Other than id and name attributes, it can be also used to locate an element on the page. The HTML source code is given below:

```
<a id="link-forgot-passwd" href="https://accounts.google.com/RecoverAccount?service=mail&amp;continue=https%3A%2F%2Fmail.google.com%2Fmail%2F" class="need-help-reverse">Need help?</a>
```
(Gmail. 2014. *Gmail*.)

To identify the element, the following line of code works:

```
driver.findElement(By.className("need-help-reverse")).click();
```

Find Web Elements by Link Text

A web element can also be located by using its text displayed on the link and to do that, the By class of WebDriver provides a method named linkText().This method is very convenient because it checks all the link text on the DOM and returns all the matches it finds out. An example is displayed below:

```
<a id="link-forgot-passwd"
href="https://accounts.google.com/RecoverAccount?service=mail&amp;continue=https%3A%2F%2Fmail.go
ogle.com%2Fmail%2F" class="need-help-reverse">Need help?</a>
```
(Gmail. 2014. *Gmail.*)

```
//Click on the link

driver.findElement (By.linkText("Need help?")).click();
```

Find Web Elements by Partial Link Text

The By class of WebDriver provides a method which is partialLinkText () to locate element using a portion of text displayed on the link. This method is more practical to use because if the link text is not fixed but dynamically assigned, then a portion of the text can be used to identify the exact element on the DOM. An example given below:

```
<a id="link-forgot-passwd"
href="https://accounts.google.com/RecoverAccount?service=mail&amp;continue=https%3A%2F%2Fmail.go
ogle.com%2Fmail%2F" class="need-help-reverse">Need help?</a>
```
(Gmail. 2014. *Gmail.*)

The code to get the element on the DOM would be:

```
//get the link

driver.findElement(By.partialLinkText("Need help")).click();
```

Find Web Elements by Tag Name

This strategy is not reliable because tagName() queries the entire DOM where it can have multi instances of the same elements and if it finds any matching elements, it will return them. This method is more likely implemented in table to find out how many rows or cells in there.

```
//focus on this tab

driver.findElement(By.tagName("html")).sendKeys(Keys.chord(Keys.CONTROL,Keys.NUMPAD9));
```

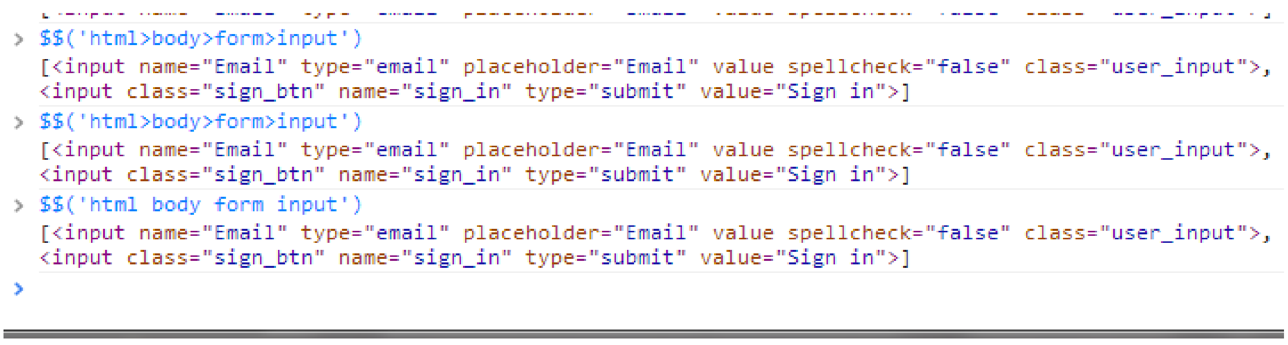
Find Web Elements by CSS Locator

The By class of Selenium WebDriver provides a method that is cssSelector() in order to locate elements. Elements can be identified in several ways such as using absolute path, relative path, class selector, id selector, attributes selector, attributes name selector etc. A point to be noted is that by using cssSelector(), it is impossible to find out a parent element using a child element unlike XPath which means that using cssSelector elements can be searched only forward not backward. To test an element in the browsers' console a function called $$() that can be used.

Using absolute path to find an element

This strategy totally depends on the hierarchy in the DOM because if the structure is changed in the DOM, the locator will fail definitely to identify the desired element on the page.

The absolute path can be verified from the browser console by writing a function in the following format $$('absolute path') for CSS Selector. If the absolute path is correct, then it can be seen from the console. Otherwise, it will display an error or return an empty list. This can be implemented in Firefox, IE, Opera and Google Chrome browsers to test if the locator can find out the element or not.



```
> $$('html>body>form>input')
  [<input name="Email" type="email" placeholder="Email" value spellcheck="false" class="user_input">,
  <input class="sign_btn" name="sign_in" type="submit" value="Sign in">]
> $$('html>body>form>input')
  [<input name="Email" type="email" placeholder="Email" value spellcheck="false" class="user_input">,
  <input class="sign_btn" name="sign_in" type="submit" value="Sign in">]
> $$('html body form input')
  [<input name="Email" type="email" placeholder="Email" value spellcheck="false" class="user_input">,
  <input class="sign_btn" name="sign_in" type="submit" value="Sign in">]
>
```

Figure 3.7.1 Locate an element using absolute XPath

In the Opera browser console, it has been tested if the absolute path can find out the desired WebElements. This (>) sign represents the child element of a parent element. This sign (>) can be replaced by a white space to represent the same objective. Furthermore, this strategy is time-consuming but it cannot correctly identify the desired elements if there are bunches of same elements on the DOM.

Using relative path to find an element

A shorter way to locate a web element directly is using a relative path. The relative path looks for the first element on the DOM. In our case, the main objective was to identify the Email Field by using a relative path. This can be seen from the browser by pressing (Ctrl+Shift+I) from the keyboard in shortcut. An example taken from the project:

```
//change to Verdana font
WebElement verdana = driver.findElement(By.xpath("//div[@*='verdana']/div[@*='J-N-Jz']"));
```

Using class selector to find an element

To find a web element on the page, the CSS class attribute can be used by specifying the type of HTML tag and then use a dot before the class attribute. This can be written as:

```
//write your pass
driver.findElement(By.cssSelector("input.Passwd")).sendKeys("hg23-driver");
```

It can also be written as:

```
//Click on the pass field
WebElement clickPass = driver.findElement(By.cssSelector(("input[class='Passwd']")));
clickPass.click();
```

Here, the class selector will look for the HTML tag <input> anywhere in the DOM whose class attribute is defined by Passwd. This can be written in other format too:

```
driver.findElement(By.cssSelector(".Passwd")).click();
```

But in this case, it will look for the class attribute named Sing in anywhere in the DOM. If there are more than one class attribute by the same name; it will return all of them.

Using ID selector to find an element

This strategy is similar to the Id locator. The ID selector can identify where a specified type of HTML tag is entered and then a hash is followed by the id attribute as shown below:

```
//Click on the sign in button
 driver.findElement(By.cssSelector("#signIn")).click();
```

This will return the particular id attribute from the DOM. A point to be noted is that if in the DOM it finds more than one occurrence of the same id attribute, it will return all of them.

Using attributes selector to find an element

The CSS locator allows locating other attributes of any element on the DOM. The code to be used to find the element “Passwd” is given below:

```
//Get the password field
driver.findElement(By.cssSelector("input[name ='Passwd']")).click();
```

Using Attribute name selector to find an element

It is known that in an HTML element there can be more than one single attributes defined. But this particular strategy is slightly different from the previous techniques by which specified attribute is used but not their attribute values. For instance, it can be seen if an element, let' say, <input> element in the DOM has a defined name or not by using the following method:

For the test purpose a Boolean not () function is used by which it can be tested if it does not match the specified criteria.

```
//Get the input element those don't have a name attribute

List <WebElement> notHaveName = driver.findElements(By.cssSelector("input:not([name])"));
```

To check if all <input> elements whose name is defined, the following method can be used:

```
//Get the input element those have a name attribute

List <WebElement> haveName = driver.findElements(By.cssSelector("input[name]"));
```

Identifying dynamically assigned attribute values to find out an element

In order to find an element in any dynamic web applications where attribute values are assigned dynamically and changing each time, a page is refreshed or reloaded. This strategy is case sensitive. For example, in such cases if the sign starts with a capital S,it will return nothing because it is case sensitive.

List <WebElement> email = driver.findElements (By.*cssSelector*("input[id^='sign']"));

The annotation ^= before the email means that if the id attribute value begins with email, then it will identify and return elements with email in the starting. This strategy is also case sensitive.

```
//Find the signIn button

List <WebElement> email = driver.findElements (By.cssSelector("input[id $='In']"));
```

$= indicates that if the id attribute ends with email, it will locate and then return elements where id attribute value ends with email in the end. This strategy is also case sensitive.

```
//Find the signIn button

List <WebElement> email = driver.findElements (By.cssSelector("input[id *='sign']"));
```

In this case, the operator *= means that wherever the id attribute matches with the value email, it should locate and return the elements.

Finding out the child elements using CSS

There is a method called nth-child () that can also be used to find out the child element of a parent element on the DOM by the following way:

```
//Find the 5th element from form
WebElement child = driver.findElement(By.cssSelector("form#gaia_loginform :nth-child(5)"));
```

It can retrieve the child elements too:

```
WebElement child = driver.findElement(By.cssSelector("form#gaia_loginform :nth-of-type(5)"));
```

The 5th element under <form> will be located and return the child element of the <form> parent element. Similarly, the first and the last element can be identified and eventually return the elements using the following ways respectively:

The first child can be found in the following ways:

```
WebElement child = driver.findElement(By.cssSelector("form#gaia_loginform :first-of-type"));
WebElement child = driver.findElement(By.cssSelector("form#gaia_loginform: first-child"));
```

The last child can be found in the following ways :

```
WebElement child = driver.findElement(By.cssSelector("form#gaia_loginform :last-child"));
WebElement child = driver.findElement(By.cssSelector("form#gaia_loginform :last-of-type"));
```

Using XPath

XPath is a query language to find out information from an XML document. The most important feature of XPath is that web elements can be searched backward or forward which means that using XPath a child element can identify its parent elements in the DOM hierarchy. Path expressions are used by XPath to select nodes or elements in any XML or HTML page. Furthermore, XPath is supported by WebDriver to locate an element using XPath query. All major browsers also support XPath. There are several kinds of nodes in XPath, for instance, attribute, element, name space and so on. XHTML in DOM is represented by HTML in a page which is actually tree based structure. WebDriver provides a method that is XPath () to identify elements on the DOM using XPath.

Some expressions are explained used by XPath to locate an element in Table 3.7.1

Table 3.7.1 Explaining XPath in brief

Expression	Meaning	Example
/	It will select from the Root element/node. Other way, if it is a direct child of a parent node/element	"html/body/div[2]/form/input[2]"
//	Element/node can be anywhere in the DOM from current node/element	"//input [contains (@id, ‘Passwd')]/../../../preceding-sibling::div[1]"
@	For selecting attributes	"//input[contains(@id,'Passwd')]/following-sibling::label"
.	Selecting current node	"//input[contains(@id,'Passwd')]/ ."
..	For selecting current element's parent node	"//input [contains (@id,'Passwd')]/parent:: form" or "//input [contains(@id,'Passwd')]/.."

Using absolute path

The main drawback of using the absolute path in XPath is that it slows down the test because XPath looks for the elements in backward and forward in the DOM. The following way it can be used to find an element in the DOM.

```
//write your signature
                driver.switchTo().frame(driver.findElement(By.xpath("/html/body/div[7]/div[3]/div/div[2]/div/div[2
]/div/div/div/div[2]/div/div/div/div/div/div/div/div/div/div/table/tbody/tr[18]/td[2]/table[2]/tbody/tr/td[2]/span/div/ta
ble/tbody/tr/td[2]/div[2]/div/iframe")));
```

An element can be tested from the browsers' console just by typing *$x ('your XPath goes here')* and then verify if the desired web element is correctly identified. However, in most major browser XPath can be retrieved just by selecting the web element from firebug in Mozilla Firefox or other browsers that have a built-in inspect element and the right clicking on the selected element and then finding copy XPath and finally copying and testing it from the console of the browsers.

Using index

By indexing the desired element can be identified on the DOM correctly if there is more <input> in this case element. Locating the password field using index can be done in the following way:

```
//get the pass field
```

```
WebElement password = driver.findElement(By.xpath("//input[11]"));
//to open the color menu
WebElement changeColor=driver.findElement(By.xpath
("//td[@*='C6']/span/div/div/div/span[2]/div/div/div/div[8]"));
```

Using relative path

The main disadvantage of using a relative path is that it cannot locate the element correctly unless there is only one <input> element, otherwise it returns all inputs found in the DOM. The following example shows how to use it:

```
WebElement password = driver.findElement(By.xpath("//input"));
```

Using attribute values with Xpath

A web element can be identified by using its attribute values like CSS Selector. Sometimes it might be the case that one attribute value is not enough to locate an element; so in this situation it is required to combine more than one attribute value for a precise match in order to locate an exact element on the DOM. Here is an example taken from Gmail [www.gmail.com] to locate password field given below:

```
WebElement button = driver.findElement(By.xpath("//input[@id='Passwd' and @name='Passwd']"));
```

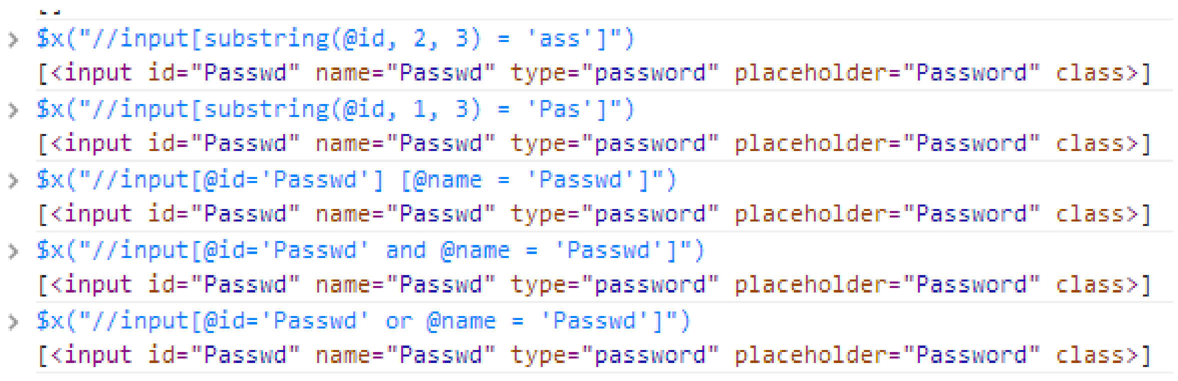
It can be written in other ways too:

```
WebElement button = driver.findElement(By.xpath("//input [@id='Passwd'] [@name='Passwd']"));
```

The following illustrates how to deal with sub-string :

```
WebElement button = driver.findElement(By.xpath("//input[substring(@id,1,3) ='Pas']"));
```

To locate an element, the substring(string, start, len) function can be used where indexing starts from 1 instead 0.In the below screen-shot, it can be seen that to match one of the attribute value, the ”or” operator is used which looks for one of the attribute value matching but the “and” operator looks for both attribute values.



```
> $x("//input[substring(@id, 2, 3) = 'ass']")
  [<input id="Passwd" name="Passwd" type="password" placeholder="Password" class>]
> $x("//input[substring(@id, 1, 3) = 'Pas']")
  [<input id="Passwd" name="Passwd" type="password" placeholder="Password" class>]
> $x("//input[@id='Passwd'] [@name = 'Passwd']")
  [<input id="Passwd" name="Passwd" type="password" placeholder="Password" class>]
> $x("//input[@id='Passwd' and @name = 'Passwd']")
  [<input id="Passwd" name="Passwd" type="password" placeholder="Password" class>]
> $x("//input[@id='Passwd' or @name = 'Passwd']")
  [<input id="Passwd" name="Passwd" type="password" placeholder="Password" class>]
```

Figure 3.7.2 Verify an element using sub-string from console of the browser

Using attributes with XPath

This technique checks if any attribute, but not the values of the attribute, is defined anywhere in the DOM for the same elements. This can be done in the following way to check the <input> name attribute if it is defined:

//Get all input having name attribute

List <WebElement> input = driver.findElements(By.*xpath*("//input [@name]"));

Using partial match

XPath provides some built-in functions to be used to locate elements on the DOM where attribute values are assigned dynamically each time a page is loaded or refreshed. In such scenario, some XPath functions such as starts-with (), ends-with () and contains () can be used to retrieve the element by matching partially like CSS Selector. This can be done in the following way.

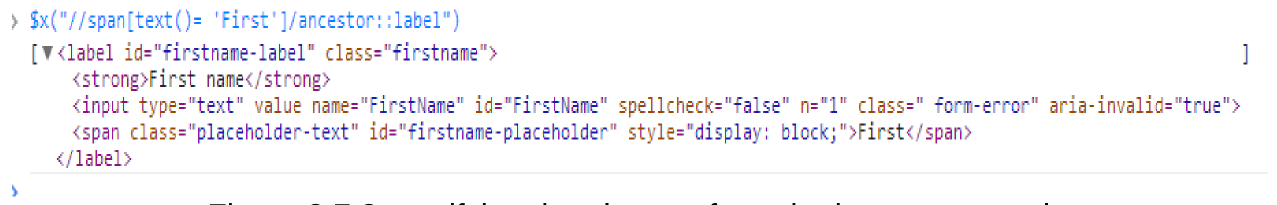
WebElement input = driver.findElement (By.*xpath*("//img [starts-with[@id,'pix_']"));

WebElement input = driver.findElement(By.*xpath* ("//img [ends-with[@id,'-pix']"));

WebElement input = driver.findElement(By.*xpath*("//img[contains[@alt,'pix']"));

Locating elements with XPath axis

With this strategy it is possible to find the relationship of elements such as child, sibling, parent, grandparent and grandchild using axes, for instance, following, descendant, following-sibling, proceeding, following-sibling, preceding, preceding-sibling etc.



```
› $x("//span[text()= 'First']/ancestor::label")
  [▼<label id="firstname-label" class="firstname">                                          ]
      <strong>First name</strong>
      <input type="text" value name="FirstName" id="FirstName" spellcheck="false" n="1" class=" form-error" aria-invalid="true">
      <span class="placeholder-text" id="firstname-placeholder" style="display: block;">First</span>
    </label>
›
```

Figure 3.7.3 verifying the element from the browser console

Table 3.7.2 demonstrates that how to identify the parent, following-sibling, preceding-sibling of an element and so on. The following result represented in Table 3.7.2 was tested with Chrome browser in this link (https://mail.google.com/mail/u/0/#settings/general).

Table 3.7.2 Techniques to find out the WebElements

Meaning	Axis	Results	Example tested in browsers' console
Before the current node it will select all sibling	preceding-sibling	Returns all <option> tag from the DOM	$x("//option[@*='fr']/preceding-sibling::option")
With the exception of ancestors, name-space, element nodes select all node those are before the current node	preceding	Returns <select>	$x("//option[@*='fr']/preceding::select")
After closing its current node it selects everything in the DOM	following	Returns all of them <select>	$x("//option[@*='fr']/following::select")
Selects all siblings after its current node	following-sibling	Return its all siblings in the DOM	$x("//option[@*='fr']/following-sibling::option")
Selects its parent	parent	Finds it parent tag name <select>	$x("//option[@*='fr']/..") or $x("//option[@*='fr']/parent ::select")
Selects parent of parent /grand parents	Grand parents	Finds it grandparent<div>	$x("//option[@*='fr']/../..")

This strategy is widely used to avoid indexing more frequently and totally straight forward. The XPath query looks for the attribute value in all elements in the DOM to check if it matches and then returns the matching element otherwise throws an error. To locate the password field:

WebElement password = driver.findElement(By.*xpath*("//input[@ *='Password']"));

One of the alternative way can be in the following way:

WebElement password = driver.findElement(By.*xpath*("//input[contains(@id,'Pass')]"));

Using some built-in XPath functions

Based on the particular text value, a search can be carried out using the text() function to check if it contains the exact text value. The function called contains() will check the text. If it matches, it will return the element.

```
//open the drop down menu

driver.findElement(By.xpath("//td[@*='C6']/span/div/div/div/span[2]/div/div/div/div[@command='+fontName']/d
iv/div/div[contains(text(),'Sans Serif')]")).click();
```

It can also be written in the following way if the exact text is known.

```
//open the drop down menu

driver.findElement(By.xpath("//td[@*='C6']/span/div/div/div/span[2]/div/div/div/div[@command='+fontName']/d
iv/div/div[.='Sans Serif']")).click();
```

In the proceeding example, it is worth noting how trimming problems can be solved to locate an element on the DOM. It can be seen that text() displays an error if there is any space before or after the text; to solve this problem, the normalize-space() function is used.

```
// dealing with trimming

WebElement title = driver.findElement(By.xpath ("//title[normalize-space(text() ='Finding the locator')]"));
```

4 WEBDRIVER AND ITS FEATURES

In this thesis, two most popular drivers, FirefoxDriver and ChromeDriver, were used.

4.1 FirefoxDriver

A Firefox plug-in is used to control the Firefox browser. Literally, Driver itself works as an extension in order to control the Firefox browser. XPCOM, a framework of Mozilla, is used by FirefoxDriver to execute the commands those are sent by the language bindings. To communicate with the FirefoxDriver, language bindings connect over a socket and send commands. The socket is called locking port that is bound to a port; the port typically would be 755 (Avasarala, 2014.p-79) or 756 (This port was shown while discovering an error during the project work).Particularly, on that port FirefoxDriver only allows at a time one instance of Firefox to listen. This is why; it is called a locking port. Once that socket is established, the Java client binding (as Java was used in my project) starts sending the commands in a serialized JSON format to the Firefox extensions. The JSON format has the following components:

- Context: current frame or window
- Command Name: For instance, dragAndDrop, SendKeys.
- Parameters: Sometimes can be empty, or text need to be typed.
- Element ID: ID is used to perform the action on it.

Earlier, this serialized JSON is actually sent over the socket or wire established to the FirefoxDriver. This is why; it works on the JSON wire Protocol. Once FirefoxDriver receives the commands from the Java client language bindings, it starts deserializes the JSON. After that, in the FirefoxDriver prototype, they are looked up while the commands are interpreted. The responses are sent back to the client over the socket after the execution. Those responses are also JSON that has methodName, Context, isError and ResponseText. (Avasarala, 2014.p-80).The main goal of using JSON is to transfer data primarily between a client and a server on the web. When an object's data is converted to JSON format is called serialization. On the other hand, if the JSON formatted data is converted to an object, this data is called de-serialization. The Firefox driver is faster than the IE driver but slower than the HtmlUnitDriver and it runs in the real browser. Test can be run on cross platform using Firefox driver and it supports JavaScript. This is how this driver works. To test script in Firefox Browser, it is recommended to have installed the latest version of the browser in the machine. Furthermore, the test can be run on different versions of Firefox browser on the same machine.

There can be such situations where a test would be run on different versions of Firefox browser already installed on the machine; in such case Firefox Binary is required to use. Once the driver is instantiated, by default, the version of Firefox is found on the path variable, is actually launched. So, to solve such kinds of problem, Firefox binary is used. In order to use the different version of Firefox, we look at the following code snippet taken from the project. To run test in the current version of the browser in the machine:

```
Capabilities cap;
String browserName,browserVersion;
//initiate the driver instance
driver = new FirefoxDriver();
cap = ((RemoteWebDriver) driver).getCapabilities();
// print name of the browser
browserName = cap.getBrowserName();
//print version
browserVersion = cap.getVersion();
```

The following line of code runs the test in Firefox 29.0.1

To run in a different version of the same browser (Firefox Version 28.0) .

```
String p = "C:\\Program Files (x86)\\Mozilla Firefox\\firefox.exe" ;
File f = new File(p);
FirefoxBinary getBinary = new FirefoxBinary(f);
FirefoxProfile createProfile = new FirefoxProfile();
FirefoxDriver driver = new FirefoxDriver(getBinary,createProfile);
```

Firefox Profile

A Firefox browser uses a folder that is known as Firefox profile to store users bookmarks settings, passwords as well as all other data. Any number of profiles can be created by a Firefox user with different custom settings and can use it accordingly.

To create, edit or delete any Firefox profile in Windows machine

Press Windows + R and then type

```
firefox.exe -p
```

And then hit enter to see the Firefox profile. To see the profile folder from the browser, open the browser and then in the address bar about: support or navigate to Help from the browser menu; then choose Troubleshooting Information.

Once an instance of Firefox Driver is created, a temporary created profile is used by the WebDriver. By default each time when a test is run, it creates an anonymous profile.

To run the test in the default Firefox browser each time, the following code solves this issue:

```
// choose a Firefox profile of your own

// get the path to the custom profile

String path
="C://Users//bhordupur//AppData//Roaming//Mozilla//Firefox//Profiles//vqxoczvi.thesis.web.driver";

FirefoxProfile customProfile = new FirefoxProfile(new File(path));

//initiate the driver instance

 driver = new FirefoxDriver(customProfile);
```

To retrieve the profile information, the FirefoxProfile class provide a method named toJson() in order to export the profile information.

Dealing with Frozen Preferences

A Firefox user is able to set any value according to Mozilla. These values are stored in prefs.js file. Those values can be overridden by a Firefox user according to user choice. user.js file is used to overwrites the default preferences by the FirefoxDriver. If any new preferences are added driver writes it to the user.js file and accordingly it behaves. So all newly added values in user.js file gets the preference over all other values are set for that specific preference. The preference files can be seen from the following path: C:\Users\bhordupur\AppData\Roaming\Mozilla\Firefox\Profiles\3uq2o2j4.default

By default a set of preferences are set by FirefoxDriver though any of the preferences are not set by the users and, therefore, it still launches the browser. However, all the preferences can be seen from user.js file. Most of the preferences cannot be changed as Firefox does not allow the tester to change and Firefox consider those as frozen preferences but some of them can be changed. If a tester changes the value, it will display an IllegalArgumentException error which says that preference may not be overridden. Firefox set the preference value, for example, user_pref("browser.shell.checkDefaultBrowser", false); otherwise when the test runs, it will show an alert saying that if the user wants to change the browser to be the default one. In this case the tester should deal with the pop up while running the test. This is why, FirefoxDriver does not let alter to be the preference values in most cases.

4.2 ChromeDriver

The way ChromeDriver works is similar to IEDriver to some extent. Mainly, it contains three components to it; the client language bindings is the first one, The browser itself is the second one, and the last one driver Chrome Driver Server which lies in between the chrome browser and the client language bindings. The port would be 9515 for ChromeDriver Server (Avasarala, 2014.p-105).ChromeDriver also uses JSONWireProtocol like IEDriver in order to communicate with the CDS.

ChromeDriver is said to be working slower than HtmlUnitDriver. It has the capability to support JavaScript. V8 is the JavaScript engine of Chrome whereas Safari uses Nitro engine. This is why the execution of JavaScript might vary. A web-kit-based browser is Chrome; so it might allow a site to verify that it works also in Safari. CSS3 and XPath are fully supported by ChromeDriver. Chromium Project itself maintains Chrome Driver. It works in the Chrome browser through Chrome binary that can be found on the Chromium project's download page. To run test in Chrome browser, it is mandatory to have installed both the Chrome Driver and the latest version of Chrome browser. The test script commands are serialized into JSON format and sent them to the CDS over the wire. The Chrome browser is controlled by the automation proxy framework of Chrome that is used by the Chrome Server.

The following code illustrates how it was used in the project:

```
String path = "C:\\Java_Work\\ThesisWork\\ChromeDriver\\chromedriver_2.8.exe";
System.setProperty("webdriver.chrome.driver", path);
_driver = new ChromeDriver();
_driver.navigate().to("http://google.com");
```

Notice that in the preceding code CDS was specified using System.proterty().Before executing any test commands, ChromeDriver first launches its server. Afterwards, a driver instance is created. It has been noticed that while executing the same test several time, the port has been changed randomly. This also happens with the IE Server where the port is being changed randomly. To stop that, WebDriver itself provides a class called ChromeDriverService. The following lines of code would prevent from changing the port:

```
try{

String path =
"C:\\Java_Work\\ThesisWork\\ChromeDriver\\chromedriver_2.8.exe";

System.setProperty("webdriver.chrome.driver", path);

ChromeDriverService.Builder googleChrome = new         ChromeDriverService.Builder();

//get the location

File file = new File(path);

//use port 2014

ChromeDriverService service =
googleChrome.usingDriverExecutable(file).usingPort(2014).build();

// start the service

service.start();

//instantiate driver

_driver = new ChromeDriver(service);

//navigate to google

_driver.navigate().to("http://google.com");

}catch(Exception e){

e.printStackTrace();

}
```

The same method can also be applied in IE to prevent from changing the port number randomly.

In the Chrome browser, it is also possible to add Chrome extensions, arguments, and even binaries to the browser using ChromeOptions.

The main purpose for choosing Firefox and Chrome, is that these two browsers are the most popular to the end users to surf on the web. The automation testing is implemented in Gmail using these two browsers.

4.3 Web Driver API

WebDriver provides a user friendly API so that it is easy to understand, explore and easier to use. However, It is not confined to any specific test framework, therefore it can be used well equally, for example, in unit testing or from a plain “main” method. This chapter will introduce WebDriver API with some examples that were taken from the project.

4.3.1 Explore Navigation

We already know that WebDriver can communicate with individual web browsers natively. Thus, it can control the browser better, not just on the page, but in the web browser itself. Navigation is one of the features of Selenium WebDriver by which a software tester can work with the browsers Back, Forward, and Refresh controls to navigate between pages or sometimes between multiple applications. WebDriver provides a method for this purpose that is navigate() .The API syntax of its is given below :

WebDriver .Navigation navigate()

An example taken here from the project:

```
//navigate to Google
driver.navigate().to("http://google.com");
```

To refresh a page the following piece of code would work:

```
//refresh the page
driver.navigate().refresh();
```

4.3.2 Handling cookies

A cookie can be used to reduce the total execution time for a test suite by skipping signing in for each test suite. To do that, we sign in for just one time and in a file write all the cookies of that specific domain. Cookies can be then loaded from the file and add it to the driver from the next log in onwards .WebDriver provides a method to fetch all of the cookies information that are loaded from a web page.

Driver.manage().getCookies()

Notices that the preceding line of code will return all cookies information that are stored are by the web page in the current session.

After loading the cookie data information from the file, and add the cookies information to the driver by the following method:

```
Cookie cookieName =new Cookie(ckName,ckValue,ckDomain,ckPath,ckExpiry,isSecure);
driver.manage().addCookie(cookieName)
```

4.4 Advance usages of WebDriver

While running the test in Firefox we came across such situations where the tests failed displaying an error, i.e., WebElement could not locate the element for various reasons. This happened because of the slow network where it takes lots of time to load the application; the second reason was that web server could not serve the page quickly due to resources constraints. Apparently, it was noticed that during the test, WebElement was not loaded on the page by the time the test script tried to identify it. To overcome such problem, the average wait time was calculated and configured so that WebDriver wait for the WebElements to load on the page before it displayed NoSuchElementException error.

Implicit wait time and explicit wait time can be implemented by which it forces to make WebDriver wait for the WebElements.

4.4.1 Implicit Wait

Implicitly wait is associated to a global timeout period and is common to all WebElements. This is used to configure the wait time of WebDriver as a whole for the AUT. Obviously, depending on the network speed test execution time can be decreased or increased. If an application is hosted on a local machine always takes less time comparing the same application hosted on a remote server would take more time to load a page. In addition to that, wait time would be configured considering the cases accordingly so that tests do not spend more time to wait for the page to be loaded or spend less time and timeout. WebDriver provides a method called manage() to handle such issues in order to set the implicitly wait time. Using implicitly wait time, maximum wait time can be set for all WebElements on the web-page. Furthermore, Before WebDriver reaches the maximum time out; it polls the DOM for the WebElements for a certain period of time on the page. However, the default timeout is set at 0.If a negative timeout is set, page load would be indefinite. Once set, it is set for the life of its object instance. In case of searching for multiple elements, WebDriver polls the DOM until it finds at least one element or the timeout expires. If the locator is used by XPath which is considered to be a slower strategy, it is better to increase implicitly wait timeout judiciously due to its adverse effect.

```
//wait for 5 secs
driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
//change the theme
driver.findElement(By.xpath("//div[@*='MOa']/div[2]/div[@*='Gpa'][contains(text(),'Change theme')]")).click();
```

In the above code, notice that maximum timeout is set 5 seconds in the browser to wait for the WebElements since it is not present immediately in the page. If it cannot load the elements within the given period of time it throws a NoSuchElementException error; otherwise it proceeds further with rest of the test.

4.4.2 Explicitly Wait

Explicitly wait time is configured to look for only a particular WebElement in the DOM before the test proceed further. On the other hand, Thread.sleep() sets the condition to wait for the specified period of time which makes the test much slower. The main goal of using explicitly wait is to execute the automation testing faster. The following code was taken from the test:

```
 //wait for 2 secs
Thread.sleep(2000);
//stop resharing
driver.findElement(By.cssSelector("div.Dp div")).click();
```

One of the ways, it can be accomplished with ExpectedConditions. Below it is shown taken from the real test:

```
//wait for the element 35 secs max
WebDriverWait wait = new WebDriverWait(driver, 35);
WebElement elementForLogOut =
wait.until(ExpectedConditions.elementToBeClickable(By.xpath("//*[@id='gb']/div[1]/div[1]/div[2]/div[5]/div[1]/a/
span")));
elementForLogOut.click();
```

Here notice in the above code snippet that driver waits maximum 35 seconds before it throws a TimeoutException error; else it continues with the next step. In every 500 milliseconds, WebDriver calls by default the ExpectedConditions until it returns the WebElement successfully. However, a successful return is always Boolean true or not null values for all ExpectedConditions types. ExpectedCondition is an interface used to implement the conditional wait for a specific WebElement. Thus, implicitly wait is overridden exclusively for the elements.

4.4.3 Taking a screen shot

One of the most useful capabilities of the WebDriver is to take a screen shot when it fails during the execution time and save it to a safe location so that it can be seen later. By default screen shot capability is enabled in all major browsers. By the way, TakesScreenshot is an interface and is implemented by Firefox, Opera, Chrome, IE Driver and so on. In three variant formats, WebDriver provides taking screen shot such as BASE64, FILE and BYTES (raw data).If FILE format is chosen, a (dot) png file is used to write the data into it, once the JVM is killed automatically after the session file is deleted unless it is saved in a local or a remote machine.

```
//take and save screen shot
File sourceFile = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(sourceFile, new File("C:\\Users\\Public\\Pictures\\testStartBrowser.png"));
System.out.println("Absolute path is :" + sourceFile.getAbsolutePath());
```

Notice that preceding code demonstrates screen shot will be stored to the local machine if it fails during the run time as a file format.

In Firefox Driver, getScreenshotAs() takes the whole page as a screen shot but in Chrome Driver only takes the visible portion.(Avasarala, 2014 ,p64)

4.4.4 Drag and Drop

The dragAndDrop() method is similar to the dragAndDropBy() method. The main difference between these two methods is that Source element is moved to the Target element whereas in dragAndDropBy() method offset used instead. A code snippet is given:

```
//drag yellow-bang to not in use from in use
WebElement drag = driver.findElement(By.xpath("//td[@*='Ut']/div[@*='sB']/div/span[@*='orange-
star']/img"));
//print CSS info
System.out.println(" the height :" + drag.getCssValue("height"));
System.out.println("the width : " +drag.getCssValue("width"));
//element is dropped on the below element
WebElement drop = driver.findElement(By.xpath("//td[@*='Ut']/div[@*='qA']"));
new Actions(driver).dragAndDrop(drag,drop).perform();
```

It can be seen that drag element would be dropped to the drop element using Actions class.

4.4.5 Moving between Windows and Frames

While working with the automation testing in Gmail, we came across such situations where multiple frames and windows to be handled. To do that, WebDriver provides a method called switchTo() to switch between frame and window or window to window or frame to frame in the same browser. This could be handled by WebDriver when a link opens in a new tab or window or a new frame. The following code shows how to move from window to a frame.

```
//still inside parent window
driver.findElement(By.xpath("//a[@*='Share']/div[2]")).click();
//wait for 5 secs
Thread.sleep(5000);
//move to iframe
driver.switchTo().frame(driver.findElement(By.cssSelector("div#gbsfw iframe")));
//send ur text
driver.findElement(By.xpath("//div[@*='Rd']/div[2]/div[2]")).sendKeys("Come To The Thesis Seminar)");
```

The following code shows how to come back to the window from frame:

To go back to the window from frame, the following line of code is used:

```
driver.switchTo().defaultContent();
```

The following code shows how to move from a parent window to a child window:

```
//get the parent window
String parentWindow = driver.getWindowHandle();
//click compose button
driver.findElement(By.xpath("//div[@*='z0']/div[contains(text(),'COMPOSE')]")).click();
//now get the child window
for (String childWindow : driver.getWindowHandles()){
//switch to child window
            driver.switchTo().window(childWindow);
            }
```

```
System.out.println("You have got  "+ driver.getWindowHandles().size() + " Windwos");
//get rid of this window
driver.findElement(By.xpath("//div[@*='aoI']/div/div[@*='Pop-in']")).click();
//back to the parent window
driver.switchTo().window(parentWindow);
```

The following code shows how to move from frame to frame.There are three ways to move from one frame to another frame that comes with WebDriver.

driver.switchTo().frame(int index)

Using the preceding piece of code an index should be used to locate the target frame in the DOM. WebDriver by default indexes the first iframe encountered in the DOM as 0.This becomes handy when a web application, for example, Gmail having bunches of iframe used in a page.

driver.switchTo().frame(String nameOrId)

if the iframe having a static or a dynamic id or a name or even a title this piece of code can be used to locate the desired iframe in the DOM.

driver.switchTo().frame(WebElement frameElement)

This piece of code is used more to locate a target iframe. This can be used when a tester is not sure about the indexing of the target iframe or iframe does not have an Id, name and title attributes at all. An example is taken below from the project:

```
//move to iframe
driver.switchTo().frame(driver.findElement(By.cssSelector("div#gbsfw iframe")));
//send ur text
driver.findElement(By.xpath("//div[@*='Rd']/div[2]/div[2]")).sendKeys("Come To The Thesis Seminar)");
//create an event
driver.findElement(By.xpath("//div[contains(text(),'Event')]")).click();
//wait for 5 secs
driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
//change the theme
driver.findElement(By.xpath("//div[@*='MOa']/div[2]/div[@*='Gpa'][contains(text(),'Change theme')]")).click();
driver.switchTo().defaultContent();
```

```
//get into the second iframe
driver.switchTo().frame(driver.findElement(By.xpath("/html/body/div[17]/div/iframe")));
Thread.sleep(2000);
driver.findElement(By.xpath("//span[contains(text(),'Just kids')]")).click();
```

Notice that to move from the first frame to the second frame defaultContent() method is used; otherwise WebDriver throws an error saying that could not locate the iframe/element.

4.4.6 Keyboard Interactions

By using the KeyBoard interface of WebDriver, three different actions can be performed on WebElements such as keyUP, KeyDown, and sendKeys actions and each of them having two overloaded methods. On the WebElement, one of the methods executes the action directly, and the other one just executes the method to the irrespective context of it.KeyDown () method simulates the action to press and hold a key [Shift, Alt, and Ctrl keys] and the keyUp() method releases the key that is pressed already by the keyDown() method. An Example is given below:

```
//get the web element
WebElement rightClick = driver.findElement(By.xpath("//*[@id='rso']/div[1]/li[2]/div/h3/a"));
//get the keyboard interactions
new Actions(driver).keyDown(Keys.CONTROL)
.click(rightClick).keyUp(Keys.CONTROL).build().perform();
```

Notice that KeyDown() is implemented on rightClick WebElement to perform an action using KeyBoard.

4.4.7 Mouse based Interactions

Mouse is an interface of WebDriver having more than seven different actions to perform using the Actions class. An example taken from the project given below:

```
//get the web element
WebElement rightClick = driver.findElement(By.xpath("//*[@id='rso']/div[1]/li[2]/div/h3/a"));
//perform an action on it
new Actions(driver).moveToElement(rightClick).perform();
Locatable loc = (Locatable)rightClick;
```

```
// Convert the driver as a Mouse
Mouse m = ((HasInputDevices)driver).getMouse();
m.contextClick(loc.getCoordinates());
Keyboard k = ((HasInputDevices)driver).getKeyboard(); k.sendKeys(Keys.CONTROL+"t");
```

5 TESTING AND REPORTING ITS RESULTS

The steps required to automate testing in Gmail using WebDriver described in Appendix 1.

5.1 Test Types

There are some facts that need to be considered on the web application before it is tested. For example, what part of the application would be automated, user expectations, time given for the project, and priorities were set by the project manager.

5.1.1 Static content testing

This is said to the easiest type of test, known as content test, because of the existence of a non-changing, static UI web element. Depending upon the software, the static content test might or might not require. The following criteria need to be considered for testing content test.

- If the page has a page title
- If the home page contains some logos or images expected to be at the certain part of the page
- If the page begins with heading tag <h1>,<h2> and so on
- If the page has correct text inside that header
- If the footer contains some links, for example, trademarks, company links, privacy policy etc.
- If the attribute values are static/constant ,for example, name ,Id, class and so on
- If the DOM structure does not change at all

5.1.2 Testing links

One of the common reasons is to automate an application is to check if the links are working properly. Frequently it is found that while clicking on a link either the link is broken or missing pages. In the test, it involves checking each link with a view to verifying the expected pages. However, if the links are altered randomly, or if files are relocated occasionally, link tests must be automated.

5.1.3 Function test

Function tests check all functionalities of a software to ensure that it works as specified in its functional requirements. The purpose of functional tests is to test a particular function on the web application. It involves very often a set of input fields, cancel or submit operations within a form in one or more response pages. The options for user input can be drop-down lists, check boxes, radio buttons, text-input fields or any other inputs that are supported by the web browsers. Function tests are the most important tests but they can be very complicated in most cases to automate. These tests are suitable to implement for log in, sign up ,settings changes, most complex data retrieval operations and so on.(Test Design Considerations — Selenium Documentation.)

5.1.4 Test for Dynamic WebElements

While dealing with Gmail, it has been noticed that elements' attribute values are being changed randomly after a while within the page. These values are generated dynamically using AJAX that varies in each instance on the DOM. For example, when attributes values for Id and class are assigned to an element that is generated dynamically, the same element is having a different attribute values once the page is reloaded or   after a few seconds attribute values are changed even though page is not reloaded yet.

5.1.5 Ajax test

A web browser technology is AJAX that stands for Asynchronous JavaScript and XML. AJAX is a set of interrelated techniques for web application that is used on the client side for creating asynchronous web applications. Generally, AJAX uses XHTML for content and CSS for presentation, as well as DOM and JavaScript to display dynamic content on the web page. It helps to create a faster, more intuitive and better web application. It exchanges small amounts of data with a server behind the scenes and then updates a portion or part of a web page or a single element itself without reloading the entire page. For example, a submit button is pressed in an AJAX data driven web application, JavaScript is used for a request to the server, and then server interprets the requested results and send it back to the client .XML format is used to retrieve the data. To locate AJAX elements, WebDriver is mainly designed where Selenium RC had limitations.

5.1.6 Validating results

This project used TestNG which can use flags when a test suite fails. TestNG also provides a class named Assert that can be used to terminate a test suite if the test does not pass. The disadvantage of using this is that it never checks the rest which were not performed at all. Therefore, information cannot be retrieved on the other status. The focus of using Assert is to get the immediate feedback.

5.2 Test design and consideration

The test automation was implemented in Google Mail (Gmail) against two of the most modern browsers out there in the market .The whole test was designed as four test cases to reduce the execution time of which each test case performs some predefined tasks. To carry out the common functions that are required by all tests are shared so that duplication can be avoided in each test script. The sequence of the test methods were considered within the test suite because of the targeted areas of the Gmail application for test automation. All the test cases were designed for functional testing in Gmail. In all test cases, the delay was set to static so that if the test ran in a slow network connection or due to server delay page takes time to load, it could still perform the defined tasks; otherwise, the test would fail.

The first test case is designed to log in to Gmail. The actions performed were on the WebElements to search for Gmail in Google and get the right link from the auto suggested Google values is shown in Test Case 1.Test Case 1 does not depend on any other test cases. For visual effect, the test case was slowed down by delay. The source code is given in the Appendix 6 as Test Case 1: “Log in to Gmail “. In Appendix 1, test case plan can be seen for log in to Gmail.

Test Case 2 was set for changing the general settings in Gmail. It depends on test case 1.It shows also the interactions with the elements; for example, to drag and drop elements from source to the target elements. It also shows how to deal with frames in the window.

The source code is given for Test Case 2: “Change General Settings” in Appendix 7 as well as a test case plan in Appendix 1.

In Test Case 3, it is shown how to attach a profile picture in Gmail. It also shows how to compose an e-mail by attaching a file from a local machine whereas a link is given in the e-mail message body. The source code is given for Test Case 3: “Send Auto Email And Set A Profile Picture” in Appendix 8.The steps for automating the Test Case 3 is presented in Appendix 1. This test depends upon Test Case 1.

The last case is for creating an event in Gmail and share it in Google +. Then verify if it is shared in Google +. Finally, this test concluded after successfully logging out from Gmail. The source code for test case 4 : “Share An Event In Google +” is given in Appendix 9.The test case plan is presented in Appendix 1.This test also depends on Test Case 1.

To optimize the performance of each test, it is considered to keep each test case smaller. When a test is designed to be smaller, it is easy to find out the bug in the scripts when the test case fails. On the other hand, a lengthy and large test case might fail for several reasons because it covers multiple features and it takes lots of time to analyze a failure in order to solve the problem. It is also recommended to run the tests in a workstation that works really faster with a high-speed network connection.

5.3 Test Execution with different browsers

Before executing the test, the first step was to set up the environment so that test script could be compiled and then test suite could be executed. Since while executing the test suite in automation testing; it does not require human intervention or supervision. The test execution once completed ,it should document the test results in details by which it can be determined which tests failed or passed, and time taken to accomplish the test suite as well as chronological order for executed methods in the test suite. In the project, test execution was implemented in Chrome and Firefox.

5.3.1 Test Execution with Firefox

A latest version of Firefox browser is preferred or a different version is required to have on the workstation before running a test in Firefox browser .In the test script, it is mandatory to instantiate FirefoxDriver in order to run the test on Firefox browser. The test execution was run on Firefox 32 bit (version 29.0.1) in 64 bit Windows 7 machine. In Appendix 4.0, 5.0, and 6.0 includes, the test scripts source code.

5.3.2 Test Execution with Google Chrome

The test was also run on Chrome 32 bit (version 33.0.1750.154 m) in 64 bit Windows 7 workstation. The same requirements were also applied to Chrome browser. The source code of test script for Chrome was pretty similar as the purpose for the test was to achieve the same goal. The execution time taken by each browser for all test suites is given below. These results were observed after running several times without any error the same test in each browser. The time limit ranges from 5-15 seconds less or more.

Table 5.3.2 comparison between execution time

Test Suite	Execution time in Firefox	Execution time in Chrome
1	47	37
2	107	113
3	105	114
4	154	149

It has been noticed that execution differs after each time being executed. It has never been the same. In some cases Chrome runs faster; sometimes Firefox runs faster. However, the time taken by each test case in Chrome and Firefox does not differ much.

5.4 Reporting results

In any automation testing, reporting has been the most important part, reason being that it helps to understand the user about the test suite execution results, point of failures and the reasons for being failed. When a test suite runs, it is possible to keep an eye on the test execution flow and for debugging by using Logging. After the execution of each test case, the results are reported to test logs (Hayes, 2004, p81). The elapsed time of each test is to be included so that performance problems and other related issues can be tracked .Alternatively; an error log file can be effective to provide detailed information about a test failure to diagnose the problems and possible determination of its cause. A false success of a test result indicates that a test was reported that it was successfully executed when it was not. A false success may occur for several reasons; for example, when a test accomplishes so quickly that it did not actually execute all the predefined steps appropriately. There are many ways to generate the test reports these days, for example, using TestNG, ReportNG, and JUnit or by writing code for that and so on.

In this project, we implemented TestNG that considered being more powerful and easy testing framework having some new functionalities comparing to JUnit or NUnit. By default, TestNG generates an HTML and XML types of reports after the test execution. It allows users to compile their own custom reporter and then it can be used with TestNG. In the following after analyzing, the test results for each test case TestNG specific HTML reports are given in the proceeding.

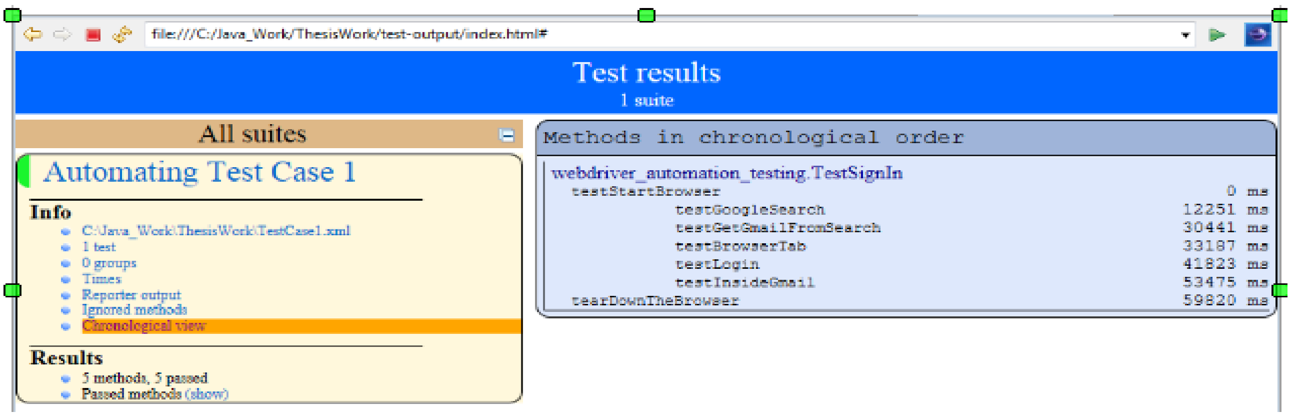
Table 5.4 Symbols used in the HTML report

Symbols	Meaning
🟦	Bullentin point
❎	Failed
✅	Passed
☒	Skipped

5.4.1 Reporting Results after executing the Test Cases in Firefox browser

The test cases results are presented here as an HTML based report. All passed test cases report is presented in the following in HTML. The prints out results for the Test case 1 from Eclipse Console are given in Appendix 3.0.

Test Case 1 HTML report for Test Case 3



file:///C:/Java_Work/ThesisWork/test-output/index.html#

Test results
1 suite

All suites

Automating Test Case 1

Info
- C:\Java_Work\ThesisWork\TestCase1.xml
- 1 test
- 0 groups
- Times
- Reporter output
- Ignored methods
- Chronological view

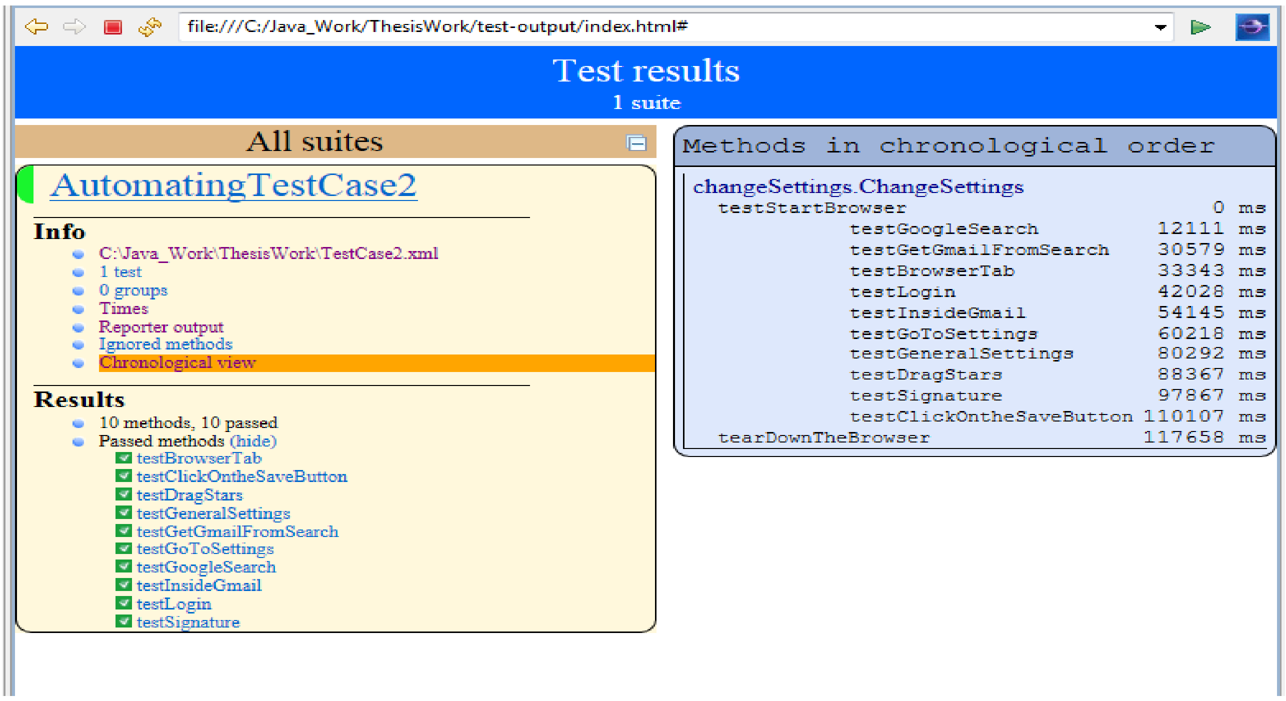
Results
- 5 methods, 5 passed
- Passed methods (show)

```
Methods in chronological order
webdriver_automation_testing.TestSignIn
  testStartBrowser                                   0 ms
           testGoogleSearch                      12251 ms
           testGetGmailFromSearch                30441 ms
           testBrowserTab                        33187 ms
           testLogin                             41823 ms
           testInsideGmail                       53475 ms
  tearDownTheBrowser                             59820 ms
```

Figure 5.4.1 Test Case 1 HTML report

Notice that the above figure demonstrates how many suites were run and their details, for example, failure and success results as well as time(in milliseconds) taken to execute each method .In the above picture ,only one test suite is run that is called "Automation Test Case 1".

Test Case 2 HTML report for Test Case 2



file:///C:/Java_Work/ThesisWork/test-output/index.html#

Test results
1 suite

All suites

AutomatingTestCase2

Info
- C:\Java_Work\ThesisWork\TestCase2.xml
- 1 test
- 0 groups
- Times
- Reporter output
- Ignored methods
- Chronological view

Results
- 10 methods, 10 passed
- Passed methods (hide)
  - ✓ testBrowserTab
  - ✓ testClickOntheSaveButton
  - ✓ testDragStars
  - ✓ testGeneralSettings
  - ✓ testGetGmailFromSearch
  - ✓ testGoToSettings
  - ✓ testGoogleSearch
  - ✓ testInsideGmail
  - ✓ testLogin
  - ✓ testSignature

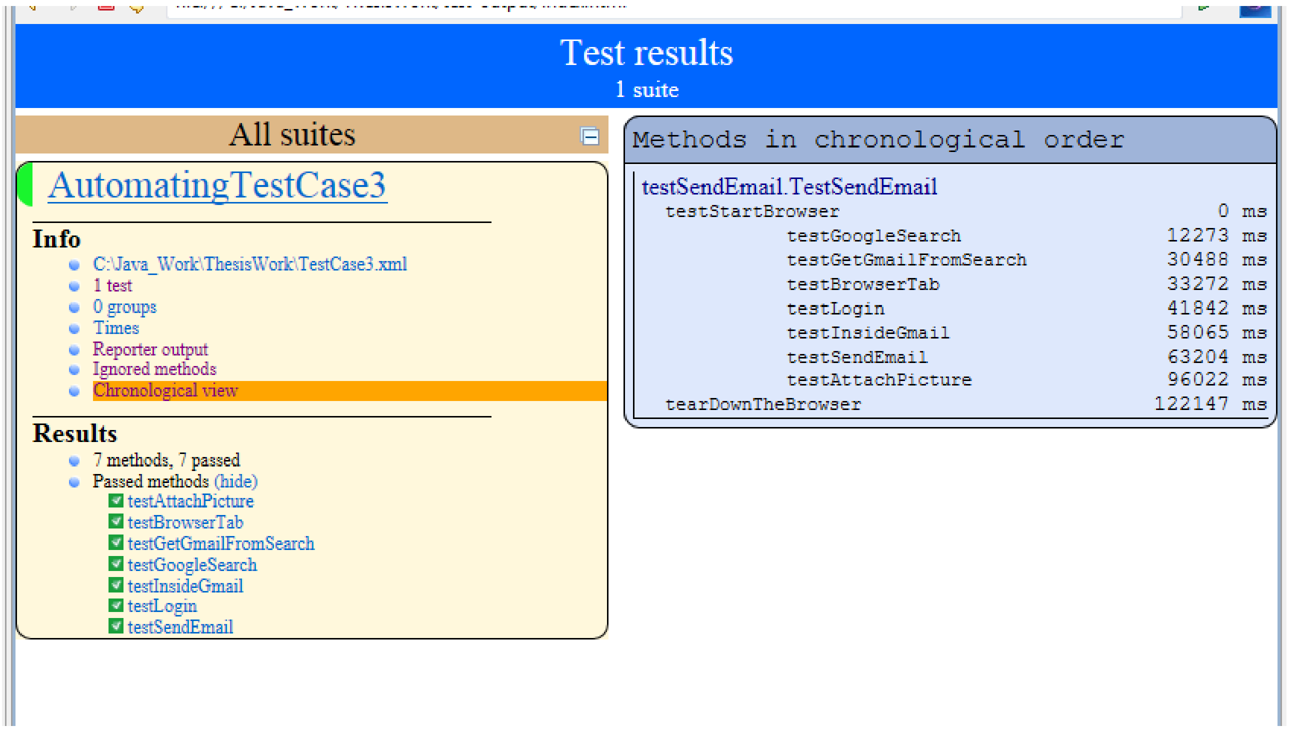
```
Methods in chronological order
changeSettings.ChangeSettings
  testStartBrowser                                0 ms
            testGoogleSearch                  12111 ms
            testGetGmailFromSearch            30579 ms
            testBrowserTab                    33343 ms
            testLogin                         42028 ms
            testInsideGmail                   54145 ms
            testGoToSettings                  60218 ms
            testGeneralSettings               80292 ms
            testDragStars                     88367 ms
            testSignature                     97867 ms
            testClickOntheSaveButton         110107 ms
  tearDownTheBrowser                         117658 ms
```

Figure 5.4.2 Test Case 2 HTML Report

Notice that in Results section, it presents the methods that are passed including the method name whereas no method is failed during the execution time. In the left bottom corner, it is shown in Results section that all methods in this suite passed successfully.

Test Case 3 HTML Report for Test Case 3

The report presented in the following is based on HTML.



Test results
1 suite

All suites

AutomatingTestCase3

Info
- C:\Java_Work\ThesisWork\TestCase3.xml
- 1 test
- 0 groups
- Times
- Reporter output
- Ignored methods
- Chronological view

Results
- 7 methods, 7 passed
- Passed methods (hide)
  - ✓ testAttachPicture
  - ✓ testBrowserTab
  - ✓ testGetGmailFromSearch
  - ✓ testGoogleSearch
  - ✓ testInsideGmail
  - ✓ testLogin
  - ✓ testSendEmail

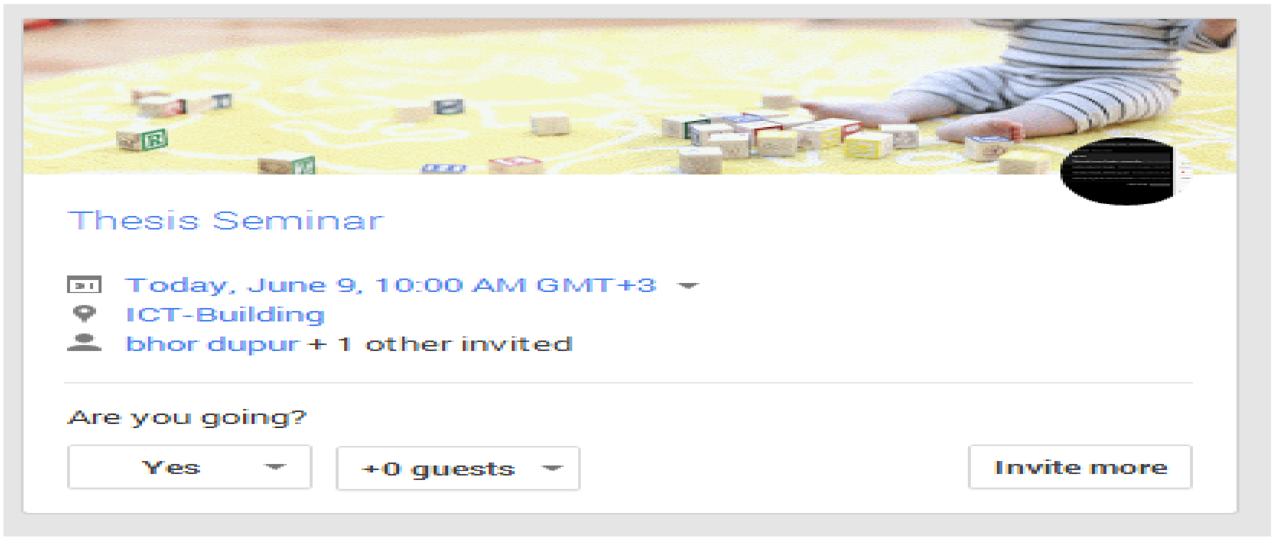
```
Methods in chronological order

testSendEmail.TestSendEmail
  testStartBrowser                                  0 ms
            testGoogleSearch                    12273 ms
            testGetGmailFromSearch              30488 ms
            testBrowserTab                      33272 ms
            testLogin                           41842 ms
            testInsideGmail                     58065 ms
            testSendEmail                       63204 ms
            testAttachPicture                   96022 ms
  tearDownTheBrowser                           122147 ms
```

Figure 5.4.3 Test Case 3 HTML Report

Notice that only one test run where 7 methods were included and all of them passed without any failure.

Test Case 4 Results for Test Case 4



[redacted]

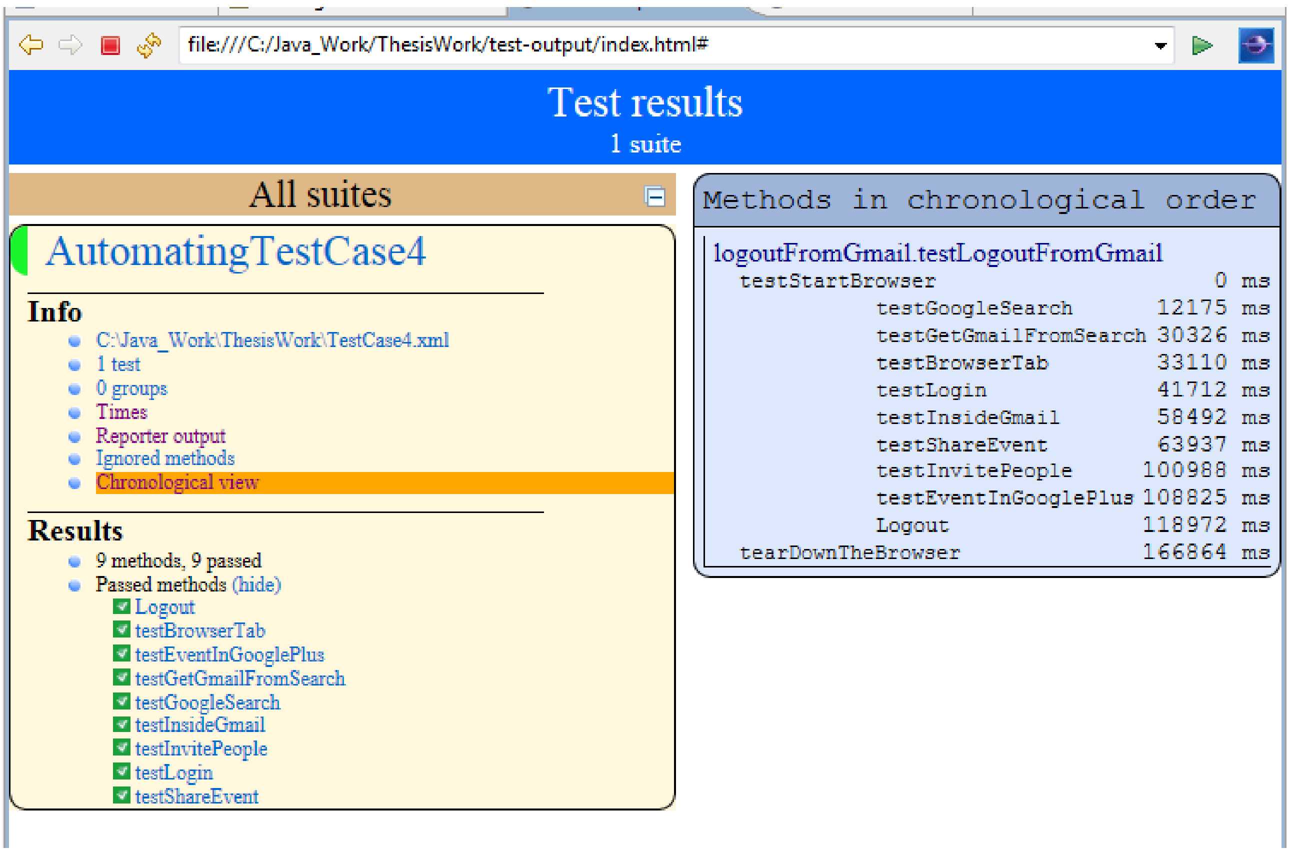
Thesis Seminar

Today, June 9, 10:00 AM GMT+3
ICT-Building
bhor dupur + 1 other invited

Are you going?

Picture 5.4.4 Taken from Google +

In Picture 5.4.4, after successfully executing the test in Chrome this picture is taken from Google + which was created and shared successfully by the test script of Test Case 4.



file:///C:/Java_Work/ThesisWork/test-output/index.html#

Test results
1 suite

All suites

AutomatingTestCase4

Info
- C:\Java_Work\ThesisWork\TestCase4.xml
- 1 test
- 0 groups
- Times
- Reporter output
- Ignored methods
- Chronological view

Results
- 9 methods, 9 passed
- Passed methods (hide)
  - ✓ Logout
  - ✓ testBrowserTab
  - ✓ testEventInGooglePlus
  - ✓ testGetGmailFromSearch
  - ✓ testGoogleSearch
  - ✓ testInsideGmail
  - ✓ testInvitePeople
  - ✓ testLogin
  - ✓ testShareEvent

Methods in chronological order

```
logoutFromGmail.testLogoutFromGmail
  testStartBrowser                          0 ms
            testGoogleSearch            12175 ms
            testGetGmailFromSearch      30326 ms
            testBrowserTab              33110 ms
            testLogin                   41712 ms
            testInsideGmail             58492 ms
            testShareEvent              63937 ms
            testInvitePeople           100988 ms
            testEventInGooglePlus      108825 ms
            Logout                     118972 ms
  tearDownTheBrowser                   166864 ms
```

Figure 5.4.4 Test Case 4 HTML Report

5.4.2 Reporting Results after executing the Test Cases in Chrome browser

The results are same in Chrome browser and Firefox browser though some of the codes in some classes are a bit different because of the browsers behavior. In the Test Case 1, it is different in case of browser's information while executing the test; else, the output is identical. The only change in the results is given below and the remainders are same.

Test Case 1

```
Starting ChromeDriver (v2.8.241075) on port 2014

WINDOWS 7 ---- is running on this machine

The system platform is : WINDOWS

The browser name is : chrome And the current verison is : 33.0.1750.154
```

Window size is by default is (1050, 708)

Newly set window dimention is (900, 700)

The rest is same with Firefox browser. On the other hand, the execution time taken by Chrome is pretty close to Firefox. There is no significance difference between them. The rest of the results can be seen from Firefox Test Cases for all test suites shown in Section 5.4.1.

5.5 Troubleshooting common problems

Throughout the project while executing the test cases after writing the code, we encountered many potential problems frequently. We had to overcome those problems to execute the test cases without any errors. These problems are listed in Appendix 2.0.

6 CONCLUSION

The main purpose of this thesis was to automate the Gmail application on multiple browsers like an end user could do to check its functionality. According to the design of the test, the goal was reached as the executed outcome was identical to the actual work. The source code can be used later if the application is changed. Probably, a minor modification is required to test its functionality on the basis of the application's requirements. If in any case it does not locate the WebElement, it is necessary to verify the element from the browsers' console to perform actions on the element or using some other tools, for instance, Firebug for Firefox.

This thesis has demonstrated how a dynamic web application can be automated to check its functionality. Although it looks fairly easy, a lot of work is required to automate Gmail because of AJAX which makes it even harder. Selenium WebDriver itself does not have its own language. It depends on other programming languages. If any of the supported programming language syntax or rule is changed, it will definitely affect on the source code given in Appendix 4 and 6.

It was easier to work with the Chrome browser when the default profile was used to run the test because Chrome has a built-in Inspect Element tool. Conversely, it was hard to deal with Firefox when the default profile was used by the WebDriver because it does not have Firebug installed in it.

Further test automation can be also carried out in IE, Chrome, Firefox and Opera with only using CSS selector where jQuery may be implemented to locate the WebElements. The execution time for each test cases run on different browsers can be compared to figure out in which browser all test cases work faster on average.

Another test automation work can be done using testNG for these four test cases whereas all test cases would be executed simultaneously in Chrome, Opera, IE and Firefox; thus, an ample amount of time could be saved.

REFERENCES

Avasarala, Satya (2014). Selenium WebDriver Practical Guide. Edition. Packt Publishing.

Burns David (2012), Selenium 2 Testing Tools Beginner's Guide. 2nd ed. Birmingham, UK: Packt Publishing Ltd.

Gundecha, Unmesh (2012). Selenium Testing Tools Cookbook. UK: Packt Publishing Ltd.

Hayes, Linda G. (2004). Automated Testing Handbook. 2nd Edition. Software Testing Inst.

Li, Kanglin (2004). Effective Software Test Automation: Developing an Automated Software Testing Tool, 1 Edition, Sybex.

Richardson, Alan (2012). Selenium Simplified, Second Edition, Compendium Developments.

ONLINE DOCUMENT

Gmail. 2014. Gmail.[ONLINE] Available at:https://accounts.google.com/ServiceLogin?sacu=1&scc=1&continue=https%3A%2F%2Fmail.google.com%2Fmail%2F&hl=fi&service=mail. [Accessed 2 May 2014].

Introduction — Selenium Documentation, 2014, Introduction — Selenium Documentation, [ONLINE] Available at: http://docs.seleniumhq.org/docs/01_introducing_selenium.jsp#to-automate-or-not-to-automate. [Accessed 30 May 2014].

Platforms Supported by Selenium. 2014. Platforms Supported by Selenium. [ONLINE] Available at: http://docs.seleniumhq.org/about/platforms.jsp. [Last Accessed 28 March 2014].

Platforms Supported by Selenium. 2014. *Platforms Supported by Selenium*. [ONLINE] Available at: http://docs.seleniumhq.org/about/platforms.jsp. [Accessed 28 March 2014].

Selenium WebDriver. 2014. Selenium WebDriver. [ONLINE] Available at: http://docs.seleniumhq.org/projects/webdriver/ [Last Accessed 28 March 2014].

Selenium 1 (Selenium RC) — Selenium Documentation. 2014. Selenium 1 (Selenium RC) — Selenium Documentation. [ONLINE] Available at: http://docs.seleniumhq.org/docs/05_selenium_rc.jsp. [Accessed 30 March 2014].

Simon Stewart. (2010). Selenium WebDriver. [ONLINE] Available at: http://aosabook.org/en/selenium.html [Last accessed 28th March 2014].

Test Design Considerations — Selenium Documentation, 2014,Test Design Considerations — Selenium Documentation. [ONLINE] Available at: http://docs.seleniumhq.org/docs/06_test_design_considerations.jsp#types-of-tests. [Accessed 01 June 2014].

APPENDICES

Appendix 1.0 Test Cases Plan and Implementation in Gmail (Google Mail)

TEST CASE	1	2	3	4
Name	Log in to Gmail	Change General Settings	Send Auto Email And Set A Profile Picture	Share An Event In Google +
ID	1	2	3	4
Test Scenario	Log in to Google Mail (Gmail)	Change general settings in Gmail	Send auto email and set a profile picture in Gmail	Create and share an event in Google +
Depends On	No Test Case	Test Case 1	Test Case 1	Test Case 1
Precondition	User already has an account in Google Mail	Successfully Passed Test Case 1	Successfully Passed Test Case 1	Successfully Passed Test Case 1
Post condition	User is logged in Gmail	General Settings has been modified	Send Email and Set a profile picture	Create an event and Share it in Google + and log-out from Gmail
Steps for the test	Open the Browser [Firefox or Chrome] Navigate to Google search page Write Gmail in the search box Navigate to the right link from the Google auto suggested links click the link to open Focus on the Gmail page and enter valid data to log-in to Gmail	Click on Settings Choose Settings From General Settings Select Language [UK English] Drag and Drop Star from In use to all stars Drag and Drop star from In Use to Not in use Click Keyboard shortcuts on Click on Signature Radio Button Write a signature Choose font Verdana ,Text Italic and Text Color Red Click on Save changes button	Click on the compose button Focus on the new window Click on the Pop In Write Email address to the To field Click on the Cc link and write an email address there Check the emails typed already Give an Email Subject Compose the email Test the link given in the email body Attach a file to the email Send the email Click on the profile	Click on the share button Type the message in the share field Click on the drop down and Disable re-shares Click on Event Click on Change theme Select the first theme under Just kids Category Write the event title Have a look in Event options Set the date and time Set the date and time to end the event Set the location of

			picture Upload a Picture from the local machine Give the profile Picture a caption name Set the profile picture	the Event Invite People Share the Event then Click on +thesis Go to Google + and verify if the Event is shared in Google + page Close the Google + page Finally Log-out and end the test
Expected Result	Post condition is fulfilled.	Passed Post-condition	Meet the criteria mentioned in Post-condition	Meet the criteria mentioned in Post-condition
Precaution	Take Screen-shot where the test case fails while executing and store it to the local machine.	Take Screen-shot where the test case fails while executing and store it to the local machine.	Take Screen-shot where the test case fails while executing and store it to the local machine	Take Screen-shot where the test case fails while executing and store it to the local machine

Appendix 2.0 Troubleshooting Common Problems

Problem	**Description**	**Solution**
Unable to connect to host 127.0.0.1 on port 7056 after 45000 ms *org.openqa.selenium.WebDriverException:* Unable to bind to locking port 7054 within	When you see these kinds of errors, you will notice that It did not open the browser even. It throws an error like this *org.openqa.selenium.firefox.NotConnectedException*	If you restart the workstation and run test case again, it will work but this problem may appear again after a while. To solve it permanently, update Java or Update to the latest Selenium WebDriver .In my case, I just updated to the Java JDK latest version; then It started working fine. If this does not resolve the problem after Java update, you can update to the latest browsers .I also upgraded to the latest version of the browsers.
3. Element is not currently visible and so may not be interacted	When you see this kind of error it throws an error like this : *org.openqa.selenium.ElementNotVisibleException* Selenium throws this because on the current window element was not visible and It was somewhere on the DOM but need to scroll down or up to see the element.	This problem can be solved by many ways but in my project I have solved it by using Actions class, Coordinates interface, and Locatable interface. It can be solved also by using Point class. These all classes and interfaces come with WebDriver itself .We just need to import them in the project when required. It can be solved by using JavaScript.
4. Offset within element cannot be scrolled into view: (0,0)	It throws this error *orq.openqa.selenium.interactions.MoveTargetOutOfBoundsException* whenever a visible element you try to find out by scrolling down or up in the window	Do not use any kind of code that scroll the element. This element already visible in the current Window.
5. Element was not in a form so could not submit	It throws an error like the following org.openqa.selenium.NoSuchElementException	The technique being implemented to find the element from the DOM was not correct. In my case XPath, so I verified the element from the browser console but I found that it finds the element in the DOM. However, It still might not work; this was absurd but it was a fact. Try using absolute path at the end if nothing works. I found in many scenarios where it behaves like strange. One of the solutions can be done by using Explicitly Wait or WebDriver Wait.

6. The given selector is either invalid or does not exist	If you see this message where it throws an error saying InvalidSelectorError, it is because the XPath was not used correctly or the ID or class value has changed dynamically in the DOM.	The solution is very tricky because in this case attributes values are changed frequently; so try to use a value from an ancestor/sibling/parent ; In most cases ancestor will help to solve this problem.
7. Error communicating with the remote browser.	The problem it throws because browser was closed by the end user before finishing the test suite.	While running the test using the browser, do not interact manually with the browser and do not close the window or tab
8. depends on nonexistent method	It throws an error *org.testng.TestNGException* because in the test suite some where there are methods dependencies.	Check the methods (method name can be seen from TestNG error) and give the correct method dependencies if you intend to use dependency.
9. Starting ChromeDriver (v2.8.241075) on port 2014 Port not available. Exiting... [0.004][SEVERE]: Could not bind socket to 0.0.0.0:2014	This is a warning not an error. This is port is already is used. More specific, chrome browser is open. So this will use another port	To use the same port before starting the tests just close the browser.

Appendix 3.0 Test Case 2 results reporting from Eclipse Console

As Test Case 2 depends on Test Case 1, both test case 1 and 2 results can be seen from this Eclipse console print out. In the following, it is being noticed that the output result was printed in the Eclipse Console.

```
[TestNG] Running:

  C:\Java_Work\ThesisWork\TestCase2.xml

WINDOWS 7   ---- is running on this machine

The system platform is : WINDOWS

The browser name is :  firefox   And the current verison is : 29.0.1

Window size is by default :(1024, 546)

Newly set window dimention is (900, 700)

The new Title is now : https://www.google.fi/?gfe_rd=cr&ei=DvqKU7eHN8aWwAOX-YGQBg

The title of the page is : Google

The URL is  : https://www.google.fi/?gfe_rd=cr&ei=DvqKU7eHN8aWwAOX-YGQBg#q=gmail

the Title is :GMAIL – SÄHKÖPOSTI GOOGLELTA

The title is from gmail : Gmail – sähköposti Googlelta

Kirjaudu sisään – Google-tilit

https://www.google.fi/?gfe_rd=cr&ei=DvqKU7eHN8aWwAOX-YGQBg#q=gmail

Second tab was closed successfully

First tab was closed successfully

https://accounts.google.com/ServiceLogin?hl=fi&continue=http://www.google.fi/%23q%3Dgmail%26biw%3D
1366%26bih%3D657

Kirjaudu sisään – Google-tilit

Page Title : Inbox (4) - thesis.web.driver@gmail.com - Gmail
```

//Test Case Starts From here

```
drop down menu opened

drop down menu disappeard

Open the drop down menu again

Settings displayed : true

You r in the Settings now.!!!

See Gmail Display Language :

Bahasa Indonesia

See Gmail Display Language :
```

```
Bahasa Melayu
See Gmail Display Language :
Català
See Gmail Display Language :
?eština
See Gmail Display Language :
Cymraeg
See Gmail Display Language :
Dansk
See Gmail Display Language :
Deutsch
See Gmail Display Language :
Eesti keel
See Gmail Display Language :
English (UK)
See Gmail Display Language :
English (US)
See Gmail Display Language :
Español
See Gmail Display Language :
Español (Latinoamérica)
See Gmail Display Language :
Euskara
See Gmail Display Language :
Filipino
See Gmail Display Language :
Français
See Gmail Display Language :
Hrvatski
See Gmail Display Language :
Italiano
total language option is shown : 58
You have Selected UK ENGLISH
```

```
 Multiple was selected :false
The settings option is sucessfully selected
 the height :1px
the width : 1px
Orange star was dropped in NOT IN USE
Signature button was clicked
You selected => verdana {font}
You selected ITALIC style
You selected red for text color
KeyBoard shortcut is off
Keyboard shortcut is on now :)
The save button is enabled : true
X and Y is respectively : (675, 391)
Get the tag name of the save changes : button
Chekc the bg color : rgba(240, 240, 240, 1)
Text is : Save Changes
Changes were saved!!Execution is done.
===============================================
AutomatingTestCase2
Total tests run: 10, Failures: 0, Skips: 0
```

//Source code for all test cases for Firefox

If you try to run this code in your workstation, make sure that you have right XPath and cssSelector because it is different even though you are using the same browser. We encountered such problems when I had two profiles for Firefox and Chrome too. We noticed that same browser is having different DOM structure in Gmail that we had to deal with. To achieve the same functionality like this thesis work, we need to verify if have to identify the right web element; else it will work just perfect.

Appendix 4.0 Open the browser (OpenTheBrowser.java)

```
/**This code will open

the browser in Firefox

*/


package browser_firefox_launch;


import java.awt.Robot;
import java.awt.event.KeyEvent;
import java.io.File;
import java.io.IOException;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.Capabilities;
import org.openqa.selenium.Dimension;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.Platform;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxBinary;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.firefox.FirefoxProfile;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.testng.Reporter;
import org.testng.annotations.AfterSuite;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;


public class OpenTheBrowser {

/**
* @param args
*/

protected WebDriver driver;

@BeforeSuite(alwaysRun=true)

public void testStartBrowser() throws InterruptedException, IOException {

try {
Capabilities cap ;
String browserName,browserVersion;

// choose a firefox profile of your own
// get the path to the custom profile
//String path
="C://Users//bhordupur//AppData//Roaming//Mozilla//Firefox//Profiles//vqxoczvi.thesis.web.driver";
//FirefoxProfile customProfile = new FirefoxProfile(new File(path));
//customProfile.
```

```
//INITIATE THE DRIVER INSTANCE
driver = new FirefoxDriver();
cap = ((RemoteWebDriver) driver).getCapabilities();

// PRINT BROWSER NAME
browserName = cap.getBrowserName();

//PRINT VERSION
browserVersion = cap.getVersion();

/**
//to test in different browser
String p = "C:\\Program Files (x86)\\Mozilla Firefox\\firefox.exe" ;
File f = new File(p);
FirefoxBinary getBinary = new FirefoxBinary(f);
FirefoxProfile createProfile = new FirefoxProfile();
FirefoxDriver driver = new FirefoxDriver(getBinary,createProfile);
**/


// DETECT THE RUNNING OS
System.out.println(System.getProperty("os.name").toUpperCase()
+ "\t ---- is running on this machine");

//PRINT THE PLATFORM
System.out.println("The system platform is : "
+ Platform.WINDOWS);

System.out.println("The browser name is :  " + browserName
+ "   And the current verison is : " + browserVersion);

//PRINT THE WINDOW SIZE
System.out.println("Window size is by default :"+ driver.manage().window().getSize());

// SET THE DIMENTION
Dimension dimen = new Dimension(900, 700);
driver.manage().window().setSize(dimen);
System.out.println("Newly set window dimention is " + dimen);

//WAIT FOR 3 SECONDS
Thread.sleep(3000);

//PRESS WINDOWS + ARROW LEFT
Robot divideWindow = new Robot();
divideWindow.keyPress(KeyEvent.VK_WINDOWS);
divideWindow.delay(100);
divideWindow.keyPress(KeyEvent.VK_LEFT);
divideWindow.delay(100);
divideWindow.keyRelease(KeyEvent.VK_LEFT);
divideWindow.delay(100);
divideWindow.keyRelease(KeyEvent.VK_WINDOWS);

//WAIT FOR 2 SECS
Thread.sleep(2000);
//MAXIMISE THE BROWSER
driver.manage().window().maximize();

//PRINT THE REPORT
Reporter.log("The browser is running");
```

```
} catch (Exception e) {

//TRACE THE ERROR
e.printStackTrace();

//TAKE AND SAVE SCREENSHOT
File sourceFile = ((TakesScreenshot) driver)
.getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(sourceFile, new File(
"C:\\Users\\Public\\Pictures\\testStartBrowser.png"));

}
}

@AfterSuite(alwaysRun=true)
public void tearDownTheBrowser(){

//CLOSE THE DRIVER
driver.close();

//QUIT THE DRIVER
driver.quit();
}


}
```

Appendix 4.1 Search In Goggle (GoogleSearch.java)

```
/** this code will search for Gmail
in Google and will get the right link from
auto suggested Google values
and will navigate to Gmail link
**/


package google_search;

import java.awt.AWTException;

import java.io.File;
import java.io.IOException;
import java.util.concurrent.TimeUnit;
import org.apache.commons.io.FileUtils;
import org.testng.Assert;
import org.testng.Reporter;
import org.testng.annotations.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.Keys;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.interactions.HasInputDevices;
import org.openqa.selenium.interactions.Keyboard;
import org.openqa.selenium.interactions.Mouse;
import org.openqa.selenium.internal.Locatable;
import browser_firefox_launch.OpenTheBrowser;

@Test(description="Google Search Page")

public class GoogleSearch extends OpenTheBrowser {


@Test(priority=2, description="Get the gmail link ")

public void testGetGmailFromSearch() throws InterruptedException, IOException{

try{
//auto suggested values from google

//get the web element
WebElement rightClick = driver.findElement(By.xpath("//*[@id='rso']/div[1]/li[2]/div/h3/a"));
//perform an action on it
new Actions(driver).moveToElement(rightClick).perform();
Locatable loc = (Locatable)rightClick;
// Convert the driver as a Mouse
Mouse m = ((HasInputDevices)driver).getMouse();
//Right Click
m.contextClick(loc.getCoordinates());
// Convert the driver as a Keyboard
Keyboard k = ((HasInputDevices)driver).getKeyboard();
//open in a new tab in the same browser
k.sendKeys(Keys.CONTROL+"t");
//wait for 2 seconds
Thread.sleep(2000);
//go to the opened tab
//new                                    Actions(driver).sendKeys(driver.findElement(By.tagName("html")),
Keys.CONTROL).sendKeys(driver.findElement(By.tagName("html")),Keys.NUMPAD2).build().perform();

driver.findElement(By.tagName("html")).sendKeys(Keys.chord(Keys.CONTROL,Keys.NUMPAD2));


//check the URL and title of the page
System.out.println("The   URL   is      :   "   +   driver.getCurrentUrl()+"\n"+   "the   Title   is   :"   +
driver.getTitle().toUpperCase());

Reporter.log("Got the correct gmail link");
}catch (Exception e) {
System.out.println(e.getMessage());
//see the screen shot
File sourceFile = ((TakesScreenshot) driver)
.getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(sourceFile, new File(
"C:\\Users\\Public\\Pictures\\testGetGmailFromSearch.png"));

}

}

@Test(priority=3, successPercentage=100,description="Switch among Tabs")
//skipping this method from the test run
//@Test(enabled = false)
public void testBrowserTab() throws AWTException, IOException{

try{
```

```
Actions a = new Actions(driver);
a.doubleClick(driver.findElement(By.name("btnG"))).build().perform();

//switch the opened tabs
//wait for 3 seconds before it happens
Thread.sleep(2000L);
WebElement dom = driver.findElement(By.tagName("html"));
Thread.sleep(2000L);
dom.sendKeys(Keys.chord(Keys.CONTROL,Keys.NUMPAD1));
//go to the last tab
Thread.sleep(2000L);
dom.sendKeys(Keys.chord(Keys.CONTROL,Keys.NUMPAD2));
Thread.sleep(2000L);
//go back to the GMail page
new Actions(driver).sendKeys(dom,Keys.NUMPAD9).build().perform();

Reporter.log("The testBrowserTab() is working fine !!!");

}
catch (Exception e) {
System.out.println(e.getMessage());
//see the screen shot
File sourceFile = ((TakesScreenshot) driver)
.getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(sourceFile, new File(
"C:\\Users\\Public\\Pictures\\testBrowserTab.png"));

}



}
//@Test(enabled=false)
@Test(invocationCount = 1, description="Test Google Search", priority=1)

public void testGoogleSearch() throws InterruptedException, IOException{
/**total wait
* 20
* seconds
*/

try {
// loading time 5 seconds
driver.manage().timeouts().implicitlyWait(5,TimeUnit.SECONDS);
//navigate to GOOGLE
driver.navigate().to("http://google.com");
//check on the right page
System.out.println("The new Title is now : "+ driver.getCurrentUrl());
//get the title of the page
System.out.println("The title of the page is : " + driver.getTitle().trim());
String s = driver.getTitle();
Assert.assertEquals("Google", s);
//wait 3 seconds
Thread.sleep(3000L);

//write GMAIL on the google search box
WebElement autoCatch =  driver.findElement(By.xpath("//input[@id='gbqfq'][@name = 'q']"));
autoCatch.sendKeys("Gmail");

Thread.sleep(2000L);
//use arrow keys [KeyUp and KeyDown] to go through the values
```

```
Actions action = new Actions(driver);
for(int i=0;i<5;i++){
action.sendKeys(autoCatch,Keys.ARROW_DOWN).perform();
Thread.sleep(1000L);
}

for(int j = 3;j>=0;j--){
action.sendKeys(autoCatch,Keys.ARROW_UP).perform();
Thread.sleep(1000L);
}

//enter GMail to get the results

action.sendKeys(autoCatch,Keys.RETURN).perform();


}catch (Exception e) {
System.out.println(e.getMessage());
//see the screen shot
File sourceFile = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);
//store it in a file on the local machine
FileUtils.copyFile(sourceFile, new File("C:\\Users\\Public\\Pictures\\testGoogleSearch.png"));

}
}

}
```

Appendix 4.2 Login in to Gmail (TestSignIn.java)

```
/**
This code will close tabs and login to Gmail
**/

package webdriver_automation_testing;

import google_search.GoogleSearch;
import java.awt.Robot;
import java.awt.event.KeyEvent;
import java.io.File;
import java.io.IOException;
import java.util.concurrent.TimeUnit;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.By;

import org.openqa.selenium.Keys;

import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.support.ui.WebDriverWait;

import org.testng.Reporter;
import org.testng.annotations.Test;
```

```
@Test(description="Login to gmail from TestSignIn class")
public class TestSignIn extends GoogleSearch  {
@Test(dependsOnMethods={"testBrowserTab"},description="Test Login method", alwaysRun=true)
public void testLogin() throws IOException,InterruptedException{

try{
By link = By.linkText("Kirjaudu sisään");
new WebDriverWait(driver,
10).until(ExpectedConditions.presenceOfElementLocated(link)).click();
Reporter.log("It has clicked the login button element");
System.out.println("The title is from gmail : "+ driver.getTitle());

//go to that tab and put
driver.findElement(By.tagName("html")).sendKeys(Keys.chord(Keys.CONTROL,Keys.NUMPAD9));
//Assert.assertTrue();
Reporter.log("Now in Gmail ready for login");
Thread.sleep(2000L);
//lets close the second tab from the browser
driver.findElement(By.tagName("html")).sendKeys(Keys.chord(Keys.CONTROL,Keys.NUMPAD2));
Robot r = new Robot();
r.keyPress(KeyEvent.VK_CONTROL);
r.keyPress(KeyEvent.VK_W);
r.keyRelease(KeyEvent.VK_CONTROL);
r.keyRelease(KeyEvent.VK_W);
System.out.println(driver.getTitle());
System.out.println(driver.getCurrentUrl());
System.out.println("Second tab was closed successfully");
r.delay(1000);
//lets close the first tab now
driver.findElement(By.tagName("html")).sendKeys(Keys.chord(Keys.CONTROL,Keys.NUMPAD1));
Robot r1 = new Robot();
r1.keyPress(KeyEvent.VK_CONTROL);
r1.keyPress(KeyEvent.VK_W);
r1.keyRelease(KeyEvent.VK_CONTROL);
r1.keyRelease(KeyEvent.VK_W);
System.out.println("First tab was closed successfully");

//focus on the last tab now
//wait for 5 seconds
Thread.sleep(3000L);
//refresh the page
//fixed the problem
driver.navigate().refresh();

System.out.println(driver.getCurrentUrl());
System.out.println(driver.getTitle());
//write your Email
driver.findElement(By.id("Email")).sendKeys("thesis.web.driver");
//write your pass
driver.findElement(By.cssSelector("input#Passwd")).sendKeys("0123-LINCOLN");

//Click on the sign in button
driver.findElement(By.cssSelector("#signIn")).click();

//wait for gmail to be loaded
driver.manage().timeouts().implicitlyWait(7, TimeUnit.SECONDS);
//navigate
driver.get("http://mail.google.com");

//focus on this tab
driver.findElement(By.tagName("html")).sendKeys(Keys.chord(Keys.CONTROL,Keys.NUMPAD9));
//implicit wait to load the page
driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);

} catch (Exception e) {
System.out.println(e);
File sourceFile = ((TakesScreenshot) driver)
.getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(sourceFile, new File(
"C:\\Users\\Public\\Pictures\\testLogin.png"));

}

}
@Test(dependsOnMethods={"testLogin"},description="After logging in to gmail", alwaysRun = true)
public void testInsideGmail() throws InterruptedException, IOException{

try {

//get the tab focused
Thread.sleep(5000L);
driver.findElement(By.tagName("html")).sendKeys(Keys.chord(Keys.CONTROL,Keys.NUMPAD9));

System.out.println("Page Title : " +driver.getTitle());
Reporter.log("logged in success!!!");

} catch (Exception e) {

e.printStackTrace();
File sourceFile = ((TakesScreenshot) driver)
.getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(sourceFile, new File(
"C:\\Users\\Public\\Pictures\\testInsideGmail.png"));
}
}

}
```

Appendix 4.3 Change General Settings (ChangeSettings.java)

```
/**
this code will navigate to settings and then change some settings in Gmail
and later will save it

**/

package changeSettings;
/**
*
* author rahman md foyzur [bhordupur]
**/

import java.awt.Robot;
import java.awt.event.KeyEvent;
import java.util.List;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;

import org.openqa.selenium.WebElement;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.interactions.internal.Coordinates;
import org.openqa.selenium.internal.Locatable;

import org.openqa.selenium.support.ui.Select;

import org.testng.Assert;
import org.testng.Reporter;
import org.testng.annotations.Test;



import webdriver_automation_testing.TestSignIn;

//SKIP THE CLASS
//@Test(enabled=false)
public class ChangeSettings extends TestSignIn {



//@Test(enabled=false)
@Test(dependsOnMethods = {"testInsideGmail"},description="Go to Settings", alwaysRun=true)
public void testGoToSettings() throws InterruptedException {

try {

//WAIT FOR A WHILE
driver.manage().timeouts().implicitlyWait(3, TimeUnit.SECONDS);
//CLICK TO THE DROP DOWN MENU
driver.findElement(By.xpath("//div[@class='aeH']/div/div/div[2]/div[3]/div")).click();
// wait for 2 seconds
Thread.sleep(2000);
System.out.println("drop down menu opened");
//disappearing the drop down
driver.findElement(By.tagName("html")).click();
System.out.println("drop down menu disappeard");
Thread.sleep(3000L);
//get back to the drop down menu
driver.findElement(By.xpath("//div[@class='aeH']/div/div/div[2]/div[3]/div")).click();
System.out.println("Open the drop down menu again");

//IDENTIFY THE ELEMENT
WebElement settings = driver.findElement(By.xpath("//*[@id=\"ms\"]/div[contains(text(),Settings)]"));
Robot items = new Robot();
items.delay(2000);
int num =0;
//GO UNTIL SETTINGS
for(num=0;num<5;num++){
items.keyPress(KeyEvent.VK_DOWN);
//SEE THE VISUAL AFFECT
Thread.sleep(1000);
}
items.keyRelease(KeyEvent.VK_DOWN);
//CHECK IF DISPLAYED
System.out.println("Settings displayed : "+ settings.isDisplayed());
//Enter when on Settings
items.keyPress(KeyEvent.VK_ENTER);
items.keyRelease(KeyEvent.VK_ENTER);
//print that you are in Settings now
System.out.println("You r in the Settings now.!!!");
//WAIT FOR 2 SEC
Thread.sleep(4000);
//GET THE REPORT
Reporter.log("Works fine so far");

} catch (Exception e) {

// SEE THE ERROR MESSAGE
System.out.println(e.getMessage());

}

}

//@Test(enabled=false)
@Test(dependsOnMethods ={"testGeneralSettings"},description="Drag and Drop some stars",
alwaysRun=true)
public void testDragStars(){
try{

//WAIT FOR 5 SEOCNDS
driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
//CLICK ON THE ALL STARS
WebElement allStar =
driver.findElement(By.xpath("/html/body/div[7]/div[3]/div/div[2]/div/div[2]/div/div/div/div[2]/div/div/div/div/div/div
/div/div/div/div/table/tbody/tr[10]/td[2]/div/table/tbody/tr/td[2]/div/span[3]"));
new Actions(driver).moveToElement(allStar).click().perform();
//DRAG THE ORANGE-GUILLMET TO IN USE FROM NOT IN USE
WebElement sourceStar = driver.findElement(By.xpath("//td[@*='Ut']/div/div/span[@*='yellow-star']"));
//"//td[@*='Ut']/div[@*='qA']"
WebElement targetPlace =
driver.findElement(By.xpath("/html/body/div[7]/div[3]/div/div[2]/div/div[2]/div/div/div/div[2]/div/div/div/div/div/div
/div/div/div/div/table/tbody/tr[10]/td[2]/div/table/tbody/tr/td[2]/div/span[3]"));
//WAIT FOR 3 SEC
Thread.sleep(3000);
//MOVE THE YELLOW-STAR TO ALL STAR
new
Actions(driver).clickAndHold(sourceStar).moveToElement(targetPlace).release(targetPlace).build().perform()
;
//WAIT FOR 3 SECS
Thread.sleep(3000);

//DRAG YELLOW-BANG TO NOT IN USE FROM IN USE
WebElement drag = driver.findElement(By.xpath("//td[@*='Ut']/div[@*='sB']/div/span[@*='orange-
star']/img"));
//PRINT CSS INFO
System.out.println(" the height :" + drag.getCssValue("height"));
System.out.println("the width : " +drag.getCssValue("width"));
//ELEMENT WILL BE DROPPED ON THE BELOW ELEMENT
WebElement drop = driver.findElement(By.xpath("//td[@*='Ut']/div[@*='qA']"));
//new Actions(driver).clickAndHold(drag).moveByOffset(0, 18).build().perform();
new Actions(driver).dragAndDrop(drag,drop).perform();
System.out.println("Orange star was dropped in NOT IN USE");

//WAIT FOR 2 SECONDS
Thread.sleep(2000);
//PRINT THE REPORT
Reporter.log("stars were dragged");
}catch(Exception e){
e.printStackTrace();
}
}
```

```
//@Test(enabled=false)
@Test(dependsOnMethods ={"testGoToSettings"},description="Go to General Settings",alwaysRun=true)
public void  testGeneralSettings(){

try{

Select dropDown = new Select(
driver.findElement(By
.xpath("/html/body/div[7]/div[3]/div/div[2]/div/div[2]/div/div/div/div[2]/div/div/div/div/div/div/div/div/div/div/table/t
body/tr/td[2]/div/select")));

// get all the options
Thread.sleep(5000);
List<WebElement> allOptions = dropDown.getOptions();
for (WebElement al : allOptions) {

// print out the values
System.out.println("See Gmail Display Language : \n"
+ al.getText());
if("Italiano".equals(al.getText())){
break;
}
}

//count
System.out.println("total language option is shown : " + allOptions.size());
// select settings from the list
//select UK English
dropDown.selectByVisibleText("Deutsch");
Thread.sleep(1000);
dropDown.selectByIndex(8);
if(!(driver.findElement(By.xpath("//option[@*='en']")).isSelected())){
dropDown.selectByIndex(8);

}
if(driver.findElement(By.xpath("//option[@*='en']")).isSelected()){
System.out.println("You have selected US ENGLISH");
}else
System.out.println("You have Selected UK ENGLISH");
//CHEKC IF SIGLE VALUE WAS SELECTED OR NOT
System.out.println(" Multiple was selected :" + dropDown.isMultiple());
//PRINT THE MESSAGE
System.out.println("The settings option is sucessfully selected");
//PRINT THE REPORT
Reporter.log("US English was selected ");


}catch(Exception e){

e.printStackTrace();
}

}


//@Test(enabled=false)
@Test(dependsOnMethods={"testDragStars"},description="Have a signature in your email")
//let's do some styling with the text for your signature
public void testSignature(){
// get a signature for your email
try {
```

```
WebElement signature = driver.findElement(By.xpath("//td[@*='C6']/input[@value='1' and
@name='sx_sg']"));
//new Actions(driver).moveToElement(signature).click().perform();
Coordinates sCut = ((Locatable)signature).getCoordinates();
sCut.inViewPort();
signature.click();
//WAIT FOR A SECOND
Thread.sleep(2000);
System.out.println("Signature button was clicked");

//write your signature

driver.switchTo().frame(driver.findElement(By.xpath("/html/body/div[7]/div[3]/div/div[2]/div/div[2]/div/div/div/div
[2]/div/div/div/div/div/div/div/div/div/div/table/tbody/tr[18]/td[2]/table[2]/tbody/tr/td[2]/span/div/table/tbody/tr/td[2
]/div[2]/div/iframe")));
WebElement insideBody = driver.findElement(By.xpath("/html/body"));
insideBody.sendKeys("Thanks for your kind support.\nTitle :Thesis Web Driver");
Robot selectText = new Robot();
//PRESS THE KEY
selectText.keyPress(KeyEvent.VK_CONTROL);
selectText.keyPress(KeyEvent.VK_A);
//RELEASE THE KEY NOW
selectText.keyRelease(KeyEvent.VK_A);
selectText.keyRelease(KeyEvent.VK_CONTROL);
driver.switchTo().defaultContent();

//OPEN THE DROPDOWN MENU
driver.findElement(By.xpath("//td[@*='C6']/span/div/div/div/span[2]/div/div/div/div[@command='+fontName']/d
iv/div/div[contains(text(),'Sans Serif')]")).click();
//WAIT FOR 2 SEC
Thread.sleep(2000);
//CHANGE TO VERDANA FONT
WebElement verdana = driver.findElement(By.xpath("//div[@*='verdana']/div[@*='J-N-Jz']"));
new Actions(driver).moveToElement(verdana).click().perform();
System.out.println("You selected => verdana {font}");
//WAIT FOR 2 SEC
Thread.sleep(2000);
//CLICK ON ITALIC
driver.findElement(By.xpath(("//div[@*='aZ']/div/div[@command='+italic']/div/div/div"))).click();
System.out.println("You selected ITALIC style");
//CHANGE THE TEXT COLOR
//TO OPEN THE COLOR MENU
WebElement changeColor =
driver.findElement(By.xpath("//td[@*='C6']/span/div/div/div/span[2]/div/div/div/div[8]"));
changeColor.click();
Thread.sleep(2000);
//GET TEXT COLOR RED
WebElement textColor = driver.findElement(By.xpath("//td[@*='T-Kw-Jn8']/div[@*='background-color:
rgb(255, 0, 0);']"));
new Actions(driver).moveToElement(textColor).click().perform();
Thread.sleep(2000);

System.out.println("You selected red for text color");
//DESELECT THE TEXT NOW
driver.findElement(By.tagName("html")).click();
//PRINT THE REPORT
Reporter.log("Signature works so far");
} catch (Exception e) {

e.printStackTrace();
```

```
}

}


//@Test(enabled=false)
//click on the Save Changes Button
@Test(dependsOnMethods={"testSignature"},description="Save all the changes you made")
public void testClickOntheSaveButton(){
try{

//CHANGE THE KEYBOARD SHORTCUT ON
//LOCATE THE ELEMENT
//WAIT FOR 5 SEOCNDS
driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
WebElement shortCut = driver.findElement(By.xpath("//td[@*='C6']/input[@value='1' and @name='bx_hs']"));
Coordinates sCut = ((Locatable)shortCut).getCoordinates();
sCut.inViewPort();
//CHECK IF SELECTED BEFORE
if (!(shortCut.isSelected())){
System.out.println("KeyBoard shortcut is off");

}else{
System.out.println("Already selected On [KeyBoard]");
//CLICK ON IT
shortCut.click();
}

//WAIT FOR 2 SEC
Thread.sleep(2000);
//PRINT MESSAGE
System.out.println("Keyboard shortcut is on now :)");
//IDENTIFY THE ELEMENT
WebElement saveBtn =  driver.findElement(By.xpath("//div[@* ='rU']/button[contains(text(),'Save
Changes')]"));
//GET THE COORDINATE OF THE ELEMENT
Coordinates c = ((Locatable)saveBtn).getCoordinates();
//SCROLL UP TO THAT ELEMENT
c.inViewPort();
//GET THE INFO
//CHECK IF THE BUTTON IS ENABLED
System.out.println("The save button is enabled : "+saveBtn.isEnabled());
//CHECK X AND Y COORDINATE FOR THIS BUTTON
System.out.println("X and Y is respectively : "+ saveBtn.getLocation());
//CHECK THE TAG NAME
System.out.println("Get the tag name of the save changes : "+ saveBtn.getTagName());
//GET THE BUTTON SIZE
System.out.println("Chekc the bg color : " + saveBtn.getCssValue("background-color"));
//GET THE TEXT
String getText = saveBtn.getText();
System.out.println("Text is : "+ getText);
//VERIFY THE TEXT
Assert.assertEquals("Save Changes",getText);
//LET THE PAGE LOAD
driver.manage().timeouts().implicitlyWait(7, TimeUnit.SECONDS);
//THEN CLICK ON IT
saveBtn.click();
System.out.println("Changes were saved!!Execution is done.");
Thread.sleep(5000);
//PRINT THE REPORT
Reporter.log("Save button saved after clicking on it");
```

```
}catch(Exception e){
//PRINT THE ERROR MESSAGE
System.out.println(  e.getMessage());
// System.out.println("Could not save changes");
Assert.fail("Could not run from ChangeSettings");
}
}


//end of the class
}
```

Appendix 4.4 Send email and attach a profile picture (TestSendEmail.java)

```
/**Send auto email with attachment

and then attach a profile picture

**/

package testSendEmail;

import java.awt.Robot;
import java.awt.Toolkit;
import java.awt.datatransfer.StringSelection;
import java.awt.event.KeyEvent;
import java.io.File;
import java.io.IOException;
import java.util.concurrent.TimeUnit;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.By;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.interactions.Actions;
import org.testng.Reporter;
import org.testng.annotations.Test;

import webdriver_automation_testing.TestSignIn;

import changeSettings.ChangeSettings;




public class TestSendEmail extends TestSignIn {
//works fine
//@Test(enabled = false)
@Test(dependsOnMethods={"testInsideGmail"},description="Send An E-mail And Attach a file to it")
public void testSendEmail() throws InterruptedException, IOException{

try {


// LET THE PAGE LOAD
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
//GET THE PARENT WINDOW
String parentWindow = driver.getWindowHandle();
//CLICK ON THE COMPOSE BUTTON
```

```
driver.findElement(By.xpath("//div[@*='z0']/div[contains(text(),'COMPOSE')]")).click();
//NOW GET THE CHILD WINDOW
for(String childWindow : driver.getWindowHandles()){

//NOW SWITCH TO THE CHILD WINDOW
driver.switchTo().window(childWindow);
}
System.out.println("You have got  "+ driver.getWindowHandles().size() + " Windwos");
//GET RID OF THIS WINDOW
driver.findElement(By.xpath("//div[@*='aoI']/div/div[@*='Pop-in']")).click();
//GET BACK TO THE PARENT WINDOW
driver.switchTo().window(parentWindow);
System.out.println("You are now in the parent Window");
//write in to field
driver.findElement(By.xpath("//td[@*='eV']/div/div/textarea[@name='to']")).sendKeys("thesis.web.driver@gm
ail.com");
//WAIT FOR 2 SECONDS
Thread.sleep(2000);
//add cc also in the email
driver.findElement(By.xpath("//div[@*='aA6']/span/span/span[contains(text(),'Cc')]")).click();
//WAIT FOR 2 SEOCNDS TO SEE THE AFFECT
Thread.sleep(2000);
//write Cc add
driver.findElement(By.xpath("//td[@*='eV']/div/div/textarea[@*='vO'][@name='cc']")).sendKeys("rupalisomoy
@gmail.com");
//subject of the email
driver.findElement(By.xpath("//input[@name='subjectbox']")).sendKeys("WebDriver");
//WAIT FOR 1 SEC
Thread.sleep(2000);
//NOW CHECK IT OUT
driver.findElement(By.xpath("//div[starts-with(@class,'aoD')][@*='1']")).click();

//GET INTO THE IFRAME
driver.switchTo().frame(driver.findElement(By.xpath(("//td[@*='Ap']/div[2]/div/iframe"))));
driver.findElement(By.tagName("body")).sendKeys("He" +
"llo Sir;\nThis message is from a webdriver user.\n\t---Thanks From \nWebDriver\nCheck out this link ->");
Thread.sleep(2000);
//GET OUT OF THE FRAME
driver.switchTo().defaultContent();
//MOVE THE MOUSE TO THE ATTACH LINK
WebElement attachFile = driver.findElement(By.xpath("//div[@*='d6']/div[@*='Attach files']/div/div/div"));
new Actions(driver).moveToElement(attachFile).build().perform();
System.out.println( "Icon of AttachFile is displayed : "+ attachFile.isDisplayed());
//MOVE TO INSERT LINK
driver.findElement(By.xpath("//div[@*='Insert link (Ctrl-K)' and @command='+link']/div/div/div")).click();
driver.manage().timeouts().implicitlyWait(3, TimeUnit.SECONDS);
//TEXT TO BE DISPLAYED
driver.findElement(By.xpath("//input[@id ='linkdialog-text']")).sendKeys(" Turku UAS");
Thread.sleep(2000);
//WRITE THE LINK
driver.findElement(By.cssSelector("input#linkdialog-onweb-tab-input")).sendKeys("http://www.tuas.fi/en/");
//TEST THIS LINK
driver.findElement(By.xpath("//div[@*='linkdialog-onweb-tab']/div[2]/div[contains(text(),'Test this
link')]")).click();
//WAIT FOR FEW SECONDS
Thread.sleep(2000);
//CLOSE THE WINDWO
Robot closeWindow = new Robot();
closeWindow.delay(1000);
//fileToBeAttached.keyPress(KeyEvent.VK_ENTER);
//fileToBeAttached.keyRelease(KeyEvent.VK_ENTER);
```

```
fileToBeAttached.keyPress(KeyEvent.VK_CONTROL);
fileToBeAttached.keyPress(KeyEvent.VK_V);
fileToBeAttached.keyRelease(KeyEvent.VK_V);
fileToBeAttached.keyRelease(KeyEvent.VK_CONTROL);
fileToBeAttached.delay(2000);
fileToBeAttached.keyPress(KeyEvent.VK_ENTER);
fileToBeAttached.keyRelease(KeyEvent.VK_ENTER);
//WIAT FOR 7 SEC
fileToBeAttached.delay(7000);
//SEND EMAIL
driver.findElement(By.xpath("//div[@*='Send (Ctrl-Enter)'][contains(text(),'Send')]")).click();
//PARENT WINDOW
System.out.println("Bck to the parent window");
//PRINT THE REPORT
Reporter.log("Mail is sent succesfully!!!");


} catch (Exception e) {
//print the error message
e.printStackTrace();
File sourceFile = ((TakesScreenshot) driver)
.getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(sourceFile, new File(
"C:\\Users\\Public\\Pictures\\testSendEmail.png"));
System.out.println("Absolute path is :" + sourceFile.getAbsolutePath());
}



}

//ATTACH A PROFILE PIXURE
//@Test(enabled = false)
@Test(dependsOnMethods={"testSendEmail"},description="attach a profile
picture",successPercentage=100,alwaysRun=true)
public void testAttachPicture() throws IOException, InterruptedException{

//move the cursor to the element
//WAIT FOR A WHILE
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
Actions moveTo = new Actions(driver);
WebElement mouseOnIcon = driver.findElement(By.xpath("//*[@id='gb']/div[2]/div/div[2]/div[5]/div/a/span"));
moveTo.moveToElement(mouseOnIcon).click().perform();
Thread.sleep(2000L);

//catch change profile picture
driver.findElement(By.xpath("//div[@title='thesis driver']")).click();
Thread.sleep(2000L);
System.out.println("change pic was clicked successfully");
//click on select a photo from a computer
//get into the frame first
driver.switchTo().frame(driver.findElement(By.xpath("/html/body/div[20]/div/iframe")));
//click on Your photos tab
//driver.findElement(By.xpath("//div[@id ='doclist']/div/div[3]/div[2]/div/div/div/div[2]/div[2]/div/span[3]")).click();
//interact with the iframe elements
driver.findElement(By.xpath("/html/body/div[2]/div/div[3]/div[2]/div/div[2]/div/div/div/div/div[2]/div/div/div/div[4]"
)).click();

//ATTACH A FILE FROM YOUR LOCAL MACHINE

try{
```

```
StringSelection s = new
StringSelection("C:\\Users\\Public\\Pictures\\testInsideGmail.png");
Toolkit.getDefaultToolkit().getSystemClipboard().setContents(s, null);
Robot rbt = new Robot();
rbt.delay(2000);
//rbt.keyPress(KeyEvent.VK_ENTER);
//rbt.keyRelease(KeyEvent.VK_ENTER);
rbt.keyPress(KeyEvent.VK_CONTROL);
rbt.keyPress(KeyEvent.VK_V);
rbt.keyRelease(KeyEvent.VK_V);
rbt.keyRelease(KeyEvent.VK_CONTROL);
rbt.delay(2000);
rbt.keyPress(KeyEvent.VK_ENTER);
rbt.keyRelease(KeyEvent.VK_ENTER);
//let it load the picture
//wait for 7 seconds
rbt.delay(7000);


//rotate the picture to the right
driver.findElement(By.xpath(".//*[@id='editbar-rotate-right']/div/div[2]")).click();
//rotate it to the left
Thread.sleep(2000L);
driver.findElement(By.xpath(".//*[@id ='editbar-rotate-left']/div")).click();

Thread.sleep(2000);
//add a caption
driver.findElement(By.xpath("//div[@*='doclist']/div/div[3]/div[2]/div/div[2]/div/div[2]/div/div/div[2]/div/div[2]/div/
div[4]/div[contains(text(),'Add Caption')]")).click();
//write a caption
driver.findElement(By.xpath("/html/body/div[2]/div/div[8]/div/div/span/input")).sendKeys("DriverPhoto");
//enter to apply
//get the input from the keyboard
Robot pressEnter = new Robot();
pressEnter.delay(2000);
pressEnter.keyPress(KeyEvent.VK_ENTER);
pressEnter.keyRelease(KeyEvent.VK_ENTER);
//PRINT THE MESSAGE
System.out.println("LOOK CAPTION NAME IS THERE NOW -:)-");
//WAIT FOR 4 SECONDS
Thread.sleep(4000);
//set the profile photo
driver.findElement(By.xpath("/html/body/div[2]/div/div[3]/div[2]/div/div[2]/div/div[2]/div[2]/div/div/div")).click();
System.out.println("You have a profile pix in Gmail for webdriver");
//PRINT THE REPORT
Reporter.log("Profile pix has been set successfully");
//GET OUT OF IFRAME
driver.switchTo().defaultContent();
//POP UP DISAPPEAR
driver.findElement(By.tagName("html")).click();
} catch(Exception e){
System.out.println("Failed to Attach File : "+ e);
File sourceFile = ((TakesScreenshot) driver)
.getScreenshotAs(OutputType.FILE);
//PRINT THE RELATIVE PATH
System.out.println("File path is : "+ sourceFile.getCanonicalPath());
FileUtils.copyFile(sourceFile, new File(
"C:\\Users\\Public\\Pictures\\moveMouseToAttachIcon.png"));
}

}
```

```
//END OF CLASS
}
```

Appendix 4.5 Share event in Google + (ShareEvent.java)

```
/**

Create an event in Gmail and then share it Google +

**/


package shareAnEvent;

import java.awt.AWTException;
import java.awt.Robot;
import java.awt.event.KeyEvent;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.internal.Locatable;
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.interactions.internal.Coordinates;
import org.testng.Reporter;
import org.testng.annotations.Test;

import testSendEmail.TestSendEmail;
import webdriver_automation_testing.TestSignIn;



public class ShareEvent extends TestSignIn{
//@Test(enabled = false)
@Test(dependsOnMethods={"testInsideGmail"},priority=1,successPercentage=100,description="Create an
Event")
public void testShareEvent() throws InterruptedException, AWTException{

try {

driver.manage().timeouts().implicitlyWait(13, TimeUnit.SECONDS);
driver.findElement(By.xpath("//a[@*='Share']/div[2]")).click();
//WAIT FOR 5 SECS
Thread.sleep(5000);
//MOVE TO THE IFRAME
driver.switchTo().frame(driver.findElement(By.cssSelector("div#gbsfw iframe")));
//SEND UR TEXT
driver.findElement(By.xpath("//div[@*='Rd']/div[2]/div[2]")).sendKeys("Come To The Thesis Seminar)");
Thread.sleep(2000);

//STOP RESHARING
driver.findElement(By.cssSelector("div.Dp div")).click();
```

```
//WAIT FOE 2 SECS
Thread.sleep(2000);
driver.findElement(By.xpath("//div[@*='d-A-B']/div[contains(text(),'Disable reshares')]")).click();
Thread.sleep(2000);
//CREATE AN EVENT
driver.findElement(By.xpath("//div[contains(text(),'Event')]")).click();
//WAIT FOR 5 SECS
driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);

//CHANGE THE THEME
driver.findElement(By.xpath("//div[@*='MOa']/div[2]/div[@*='Gpa'][contains(text(),'Change theme')]")).click();
driver.switchTo().defaultContent();
//GET INTO THE SECOND FRAME
driver.switchTo().frame(driver.findElement(By.xpath("/html/body/div[17]/div/iframe")));
Thread.sleep(2000);
driver.findElement(By.xpath("//span[contains(text(),'Just kids')]")).click();
driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);


//GET THE SECOND THEME
//div[@*='Ed-eb-Ic']/table/tbody/tr[1]/td/div/div/img
//ABSOLUTE FIX THE PROBLEM
WebElement iTheme =
driver.findElement(By.xpath("/html/body/div[2]/div/div[2]/div[2]/div/div[2]/div/div[2]/div/div/div/table/tbody/tr/td/
div/div/img"));
System.out.println("iTheme is displayed : "+ iTheme.isDisplayed());

//CHECK THE THEME
if(!(iTheme.isSelected())){
iTheme.click();

}else
System.out.println("Theme is selected already");

//SAVE THE THEME
//"//div[contains(text(),'Set as theme')]"
//ABSOLUTE PATH IS SOLVING THE PROBLEM
WebElement saveButton =
driver.findElement(By.xpath("/html/body/div[2]/div/div[2]/div[2]/div/div[2]/div/div[2]/div[2]/div/div/div"));
System.out.println("Save Theme is displayed : "+ saveButton.isDisplayed());
System.out.println("iTheme was selected succesfully ");
saveButton.click();

driver.switchTo().defaultContent();

//MOVE BACK TO THE PREVIOUS FRAME
driver.switchTo().frame(driver.findElement(By.cssSelector("div#gbsfw iframe")));
Thread.sleep(2000);
//WRITE THE TITLE OF THE EVENT
WebElement eventTitle = driver.findElement(By.xpath("//div[@*='yk']/div[2]/div/input[@*='Event title']"));
eventTitle.sendKeys("Thesis Seminar");
Thread.sleep(1000);
//CHECK THE EVENT OPTIONS
driver.findElement(By.xpath("//div[@*='bda']/div[2]/div[2]/div[2]")).click();
Thread.sleep(2000);
//SET THE  DATE
driver.findElement(By.xpath("//div[@*='zC']/input[@*='text']")).click();
Robot setDate = new Robot();
setDate.delay(1000);
setDate.keyPress(KeyEvent.VK_CONTROL);
setDate.keyPress(KeyEvent.VK_A);
setDate.keyRelease(KeyEvent.VK_A);
setDate.keyRelease(KeyEvent.VK_CONTROL);
//CLEAR THE FIELD
setDate.keyPress(KeyEvent.VK_ENTER);
setDate.keyRelease(KeyEvent.VK_ENTER);
//SET THE DATE NOW
WebElement verifyDate = driver.findElement(By.xpath("//div[@*='zC']/input[@*='text']"));
verifyDate.sendKeys("Mon, Jun 9, 2014");
//CLICK ON THE DATE
verifyDate.click();
//SEE THE CALENDAR
Thread.sleep(2000);
//SET THE TIME
driver.findElement(By.xpath("//div[@*='Oma']/div")).click();
Thread.sleep(2000);
//END THE EVENT TIME AT 12:00 PM
WebElement setStartTime = driver.findElement(By.xpath("//div[contains(text(),'10:00')]"));
new Actions(driver).moveToElement(setStartTime).click().perform();
Thread.sleep(2000);
//SET THE END TIME
driver.findElement(By.xpath("//span[contains(text(),'Add end time')]")).click();
//OPEN THE DROPDOWN
driver.findElement(By.xpath("//div[@*='yk']/div[3]/div[5]/input")).click();
Thread.sleep(1000);
WebElement setEndTime = driver.findElement(By.xpath("//div[contains(text(),'11:00')]"));
new Actions(driver).moveToElement(setEndTime).click().perform();
//WRITE THE LOCATION
WebElement setLocation = driver.findElement(By.xpath("//div[@*='yk']/div[4]/div/input"));
setLocation.sendKeys("ICT");
Robot getLocation = new Robot();
getLocation.delay(2000);
getLocation.keyPress(KeyEvent.VK_DOWN);
getLocation.keyRelease(KeyEvent.VK_DOWN);
getLocation.keyPress(KeyEvent.VK_ENTER);
getLocation.keyRelease(KeyEvent.VK_ENTER);
getLocation.delay(1000);
//CHECK THE LOCATION IF IT IS CORRECT
setLocation.click();
} catch (Exception e) {

e.printStackTrace();
}


}
//@Test(enabled=false)
@Test(dependsOnMethods={"testShareEvent"},description="invite people",alwaysRun=true,priority=2)
public void testInvitePeople() throws AWTException, InterruptedException{
try {

//INVITE PEOPLE
WebElement invitePublic = driver.findElement(By.cssSelector("input#sbdp"));
invitePublic.sendKeys("Pub");
Robot getPublic = new Robot();
getPublic.delay(1000);
//GET PUBLIC
getPublic.keyPress(KeyEvent.VK_ENTER);
getPublic.keyRelease(KeyEvent.VK_ENTER);
//GET BHORUDPUR
invitePublic.sendKeys("bhordupur");
//WAIT FOR THE VALUE
```

```
getPublic.delay(2000);
getPublic.keyPress(KeyEvent.VK_ENTER);
getPublic.keyRelease(KeyEvent.VK_ENTER);
//GET PATRIC
invitePublic.sendKeys("patric gran");
//WAIT FOR THE VALUE
getPublic.delay(2000);
getPublic.keyPress(KeyEvent.VK_ENTER);
getPublic.keyRelease(KeyEvent.VK_ENTER);

//GET WHOEVER IN YOUR CIRCLE
invitePublic.sendKeys("cir");
//WAIT FOR THE VALUE
getPublic.delay(2000);
getPublic.keyPress(KeyEvent.VK_ENTER);
getPublic.keyRelease(KeyEvent.VK_ENTER);

Thread.sleep(2000);
//CLICK INVITE
driver.findElement(By.xpath("//td[@*='bI']/div[contains(text(),'Invite')]")).click();
driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
System.out.println("Invitation is sent ");
//PRINT THE REPORT
Reporter.log("Invitaiton was sent successfully");
} catch (Exception e) {

e.printStackTrace();
}
}

//@Test(enabled=false)
//@Test(dependsOnMethods={"testInsideGmail"})
//dependsOnMethods={"testInvitePeople"},
@Test(dependsOnMethods={"testInvitePeople"},description="share the Event",alwaysRun=true,priority=3)

public void testEventInGooglePlus(){

try{
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
/**
//div[@*=nH]/div/div[3]/div[@*='gb']/div/div/div/div/a[contains(text(),'+thesis')]")
//"/html/body/div[7]/div[3]/div/div/div[3]/div/div/div/div/div/a"
//html/body/div[7]/div[3]/div/div/div[3]/div/div[2]/div/div/div/a
//("//a[contains(text(),'+thesis')]"))
//("//*[@id='gb']/div[2]/div[1]/div[1]/div/a[contains(text(),'+thesis')]")

*/

driver.findElement(By.xpath("//a[contains(text(),'+thesis')]")).click();


//PAGE TO BE LOADED
driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
//NOW SWITCH TO THE LAST OPENED TAB
WebElement goToGooglePlus = driver.findElement(By.tagName("html"));
//GET THE SECOND TAB
goToGooglePlus.sendKeys(Keys.chord(Keys.CONTROL,Keys.NUMPAD2));
System.out.println("See the event that is shared in Google+(PLUS)");


//CLOSE THE TAB
```

```
Thread.sleep(5000);
Robot googlePlusTab = new Robot();
//WAIT
googlePlusTab.delay(2000);
googlePlusTab.keyPress(KeyEvent.VK_CONTROL);
googlePlusTab.keyPress(KeyEvent.VK_W);
googlePlusTab.keyRelease(KeyEvent.VK_CONTROL);
googlePlusTab.keyRelease(KeyEvent.VK_W);
googlePlusTab.delay(1000);
System.out.println("The Game is over");
}catch(Exception e){
//GET THE ERROR
e.printStackTrace();
}

}
//END OF CLASS
}
```

Appendix 4.6 Logout from Gmail (testLogoutFromGmail.java)

```
/**
Logout from Gmail
*/

package logoutFromGmail;

import java.io.IOException;
import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.interactions.Action;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.testng.Reporter;
import org.testng.annotations.Test;
import sharean.event.ingoogle.plus.ShareEventUsingChrome;




public class testLogoutFromGmail extends ShareEvent  {

@Test(enabled=false)
//@Test(dependsOnMethods={""},description="Signed out methods")
public void Logout() throws IOException {


try{

//WAIT FOR 3 SECS
Thread.sleep(3000);
//WAIT FOR THE ELEMENT 35 SECONDS MAX
WebDriverWait wait = new WebDriverWait(driver, 35);

WebElement elementForLogOut = wait.until(ExpectedConditions.elementToBeClickable
(By.xpath("//*[@id=\"gb\"]/div[2]/div[1]/div[2]/div[5]/div[1]/a/span")));
elementForLogOut.click();
//AGAIN WAIT FOR THE ELEMENT
Thread.sleep(3000);
//CLICK TO  BE LOGGED OUT
WebElement logOut =  driver.findElement(By.xpath("//div[@*='gb_ia']/div[2]/a[contains(text(),'Sign out')]"));
Actions clickLogOut = new Actions(driver);
clickLogOut.moveToElement(logOut).click();
Action perform = clickLogOut.build();
perform.perform();


//LET THE LOCATION IF YOU  ARE SINGED OUT
System.out.println("You ar successfully logged out from Gmail"+ " \n"+ elementForLogOut.getTagName());
System.out.println("Test went successfully and nothing failed ->");
//PRINT A MESSAGE
Reporter.log("Test is done successfully.Logged Out");

}
catch(Exception e){
//PRINT THE ERROR MESSAGE
e.printStackTrace();



}


}

}
```

Appendix 4.7 Store cookie (CookieForGmail.java)

```
/**

Store cookie

**/

package Cookie;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;

import org.openqa.selenium.By;
import org.openqa.selenium.Cookie;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;

public class CookieForGmail {

/**
* @param args
*/
public static void main(String[] args) {
```

```
// TODO Auto-generated method stub


WebDriver driver =new FirefoxDriver();
driver.navigate().to("http://www.gmail.com");
//give the email id
WebElement email = driver.findElement(By.id("Email"));
email.sendKeys("thesis.web.driver@gmail.com");
//give the password
//shortcut way
driver.findElement(By.cssSelector("input#Passwd")).sendKeys("xxxxxx");
//check on Stay signed in check box
driver.findElement(By.cssSelector("input#PersistentCookie")).click();
driver.findElement(By.cssSelector("input#Passwd")).submit();

//save the data in a file
//let the driver load from the file
File file = new File("C:\\Java_Work\\cookie.data");
try{
//Check the file path
file.getAbsolutePath();
//delete file if exist
file.delete();
file.createNewFile();
FileWriter writer = new FileWriter(file);
BufferedWriter bWriter= new BufferedWriter(writer);
//get all the cookies
for(Cookie cookie: driver.manage().getCookies()){
//write in the file cookie info
//now check if it is a secure cookie
bWriter.write(("Cookie Domain :" + cookie.getDomain() +"\n"+ "Cookie Name :"+cookie.getName()+"\n"
+"Cookie Value :"+cookie.getValue()+ "\n"+"Cookie Path :"+cookie.getPath()+"\n"+
"Cookie Expiry Date :"+ cookie.getExpiry()+cookie.isSecure()));
//write in a new line
bWriter.newLine();

}
bWriter.flush();
bWriter.close();
writer.close();

}catch(Exception e){

System.out.println(e);
}
}

}
```

Appendix 5.0 (Test Suite XML files)

XML files are also given.

TestCase1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Automating Test Case 1">
<test name="Test Case 1">
<classes>
<class name="webdriver_automation_testing.TestSignIn"/>
</classes>
</test>

</suite>
```

TestCase2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="AutomatingTestCase2">
<test name="TestCase2">

<classes>
<class name="changeSettings.ChangeSettings"/>
</classes>
</test>

</suite>
```

TestCase3.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="AutomatingTestCase3">
<test name="TestCase3">

<classes>
<class name="testSendEmail.TestSendEmail"/>
</classes>
</test>

</suite>
```

TestCase4.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="AutomatingTestCase4">
<test name="TestCase4">

<classes>
<class name="logoutFromGmail.testLogoutFromGmail"/>
</classes>
</test>
```

```
</suite>
```

/***

FOR CHROME

****/**

The source code for Chrome is literally same excluding XPath and cssSelector. To show the proof I would like to show you the first test case as if one can compare.

Appendix 6.0 Open the browser in Chrome (OpenChromeBrowser.java)

```
/** open the Chrome browser

look code is also same

**/

package browser_chrome;

import java.awt.Robot;
import java.awt.event.KeyEvent;
import java.io.File;
import java.io.IOException;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.Capabilities;
import org.openqa.selenium.Dimension;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.Platform;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeDriverService;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.testng.Reporter;
import org.testng.annotations.Test;

public class OpenChromeBrowser {

protected WebDriver _driver;
@Test
public void openGoogleChrome() throws InterruptedException,IOException{

try{
Capabilities cap ;
String browserName,browserVersion;
String path = "C:\\Java_Work\\ThesisWork\\ChromeDriver\\chromedriver_2.8.exe";
System.setProperty("webdriver.chrome.driver", path);

ChromeDriverService.Builder googleChrome = new ChromeDriverService.Builder();
//GET THE FILE LOCATION
File file = new File(path);
```

```
//USE PORT 2014
ChromeDriverService service = googleChrome.usingDriverExecutable(file).usingPort(2014).build();
//START THE SERVICE
service.start();
//INSTANTIATE DRIVER
_driver = new ChromeDriver(service);
cap = ((RemoteWebDriver) _driver).getCapabilities();

// PRINT BROWSER NAME
browserName = cap.getBrowserName();

//PRINT VERSION
browserVersion = cap.getVersion();
// DETECT THE RUNNING OS
System.out.println(System.getProperty("os.name").toUpperCase()
+ "\t ---- is running on this machine");

//PRINT THE PLATFORM
System.out.println("The system platform is : "
+ Platform.WINDOWS);

System.out.println("The browser name is :  " + browserName
+ "   And the current verison is : " + browserVersion);

//PRINT THE WINDOW SIZE
System.out.println(_driver.manage().window().getSize());

// SET THE DIMENTION
Dimension dimen = new Dimension(900, 700);
_driver.manage().window().setSize(dimen);
System.out.println(dimen);

//WAIT FOR 3 SECONDS
Thread.sleep(3000);

//PRESS WINDOWS + ARROW LEFT
Robot divideWindow = new Robot();
divideWindow.keyPress(KeyEvent.VK_WINDOWS);
divideWindow.delay(100);
divideWindow.keyPress(KeyEvent.VK_LEFT);
divideWindow.delay(100);
divideWindow.keyRelease(KeyEvent.VK_LEFT);
divideWindow.delay(100);
divideWindow.keyRelease(KeyEvent.VK_WINDOWS);

//WAIT FOR 2 SECS
Thread.sleep(2000);

//MAXIMISE THE BROWSER
_driver.manage().window().maximize();

//GO TO GOOGLE
_driver.navigate().to("http://google.com");
//PRINT THE REPORT
Reporter.log("The Chrome browser is running");

}catch(Exception e){
e.printStackTrace();
//TRACE THE ERROR
e.printStackTrace();
```

```
//TAKE AND SAVE SCREENSHOT
File sourceFile = ((TakesScreenshot) _driver)
.getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(sourceFile, new File(
"C:\\Users\\Public\\Pictures\\openGoogleChrome.png"));
}
}
}
```

Appendix 6.1 Search in Google (testGoogleSearch.java)

```
/**

Search in Google and navigate to Gmail

**/

package chrome.google.search;

import java.awt.AWTException;
import java.io.File;
import java.io.IOException;
import java.util.concurrent.TimeUnit;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.interactions.Actions;
import org.testng.Assert;
import org.testng.Reporter;
import org.testng.annotations.Test;


import browser_chrome.OpenChromeBrowser;


@Test(description="Google Search Page in Chrome")
public class testGoogleSearch extends OpenChromeBrowser  {


@Test(priority=2, description="Get the gmail link in Chrome ")

public void testGetGmailFromSearch() throws InterruptedException, IOException{

try{

//get the web element
_driver.findElement(By.xpath("//*[@id='rso']/div[1]/li[2]/div/h3/a")).click();
//check the URL and title of the page
System.out.println("The URL is  : " + _driver.getCurrentUrl()+"\n"+ "the Title is :" +
_driver.getTitle().toUpperCase());
Reporter.log("Got the correct gmail link");
}catch (Exception e) {
System.out.println(e.getMessage());
//see the screen shot
```

```
File sourceFile = ((TakesScreenshot) _driver)
.getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(sourceFile, new File(
"C:\\Users\\Public\\Pictures\\rightClickForlinkinChrome.png"));

}

}



//@Test(enabled=false)
@Test(invocationCount = 1, description="Test Google Search in Chrome", priority=1)

public void testGoogleSearchChrome() throws InterruptedException, IOException{
/**total wait
* 20
* seconds
*/

try {
// loading time 5 seconds
_driver.manage().timeouts().implicitlyWait(5,TimeUnit.SECONDS);
//navigate to GOOGLE
_driver.navigate().to("http://google.com");
//check on the right page
System.out.println("The new Title is now : "+ _driver.getCurrentUrl());
//get the title of the page
System.out.println("The title of the page is : " + _driver.getTitle().trim());
String s = _driver.getTitle();
Assert.assertEquals("Google", s);
//wait 3 seconds
Thread.sleep(3000L);

//write GMAIL on the google search box
WebElement autoCatch =  _driver.findElement(By.xpath("//input[@id='gbqfq'][@name = 'q']"));
autoCatch.sendKeys("Gmail");

Thread.sleep(2000L);
//use arrow keys [KeyUp and KeyDown] to go through the values
Actions action = new Actions(_driver);
for(int i=0;i<5;i++){
action.sendKeys(autoCatch,Keys.ARROW_DOWN).perform();
Thread.sleep(1000L);
}

for(int j = 3;j>=0;j--){
action.sendKeys(autoCatch,Keys.ARROW_UP).perform();
Thread.sleep(1000L);
}

//enter GMail to get the results

action.sendKeys(autoCatch,Keys.RETURN).perform();


}catch (Exception e) {
System.out.println(e.getMessage());
//see the screen shot
File sourceFile = ((TakesScreenshot) _driver).getScreenshotAs(OutputType.FILE);
//store it in a file on the local machine
```

```
FileUtils.copyFile(sourceFile, new
File("C:\\Users\\Public\\Pictures\\testGoogleSearchChromeBrowser.png"));

}
}

}
```

Appendix 6.2 Login to Gmail (TestChromeSignIn.java)

```
/**

Login to Gmail

**/

package chrome.google.search;

import java.io.File;
import java.io.IOException;
import java.util.concurrent.TimeUnit;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.By;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.testng.Reporter;
import org.testng.annotations.Test;

public class TestChromeSignIn extends testGoogleSearch{


@Test(dependsOnMethods={"testGetGmailFromSearch"},description="Test Login method",
alwaysRun=true)

public void testChromeLogin() throws IOException,InterruptedException{

try{

By link = By.linkText("Kirjaudu sisään");
new WebDriverWait(_driver,
10).until(ExpectedConditions.presenceOfElementLocated(link)).click();
Reporter.log("It has clicked the login button element");
System.out.println("The title is from gmail : "+ _driver.getTitle());
Reporter.log("Now in Gmail ready for login");
Thread.sleep(2000L);
//refresh the page
_driver.navigate().refresh();
System.out.println(_driver.getCurrentUrl());
System.out.println(_driver.getTitle());
//write your Email
_driver.findElement(By.id("Email")).sendKeys("thesis.web.driver");
//write your pass
_driver.findElement(By.cssSelector("input#Passwd")).sendKeys("0123-LINCOLN");

//Click on the sign in button
_driver.findElement(By.cssSelector("#signIn")).click();

//wait for gmail to be loaded
_driver.manage().timeouts().implicitlyWait(7, TimeUnit.SECONDS);

//implicit wait to load the page
_driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);

} catch (Exception e) {
System.out.println(e);
File sourceFile = ((TakesScreenshot) _driver)
.getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(sourceFile, new File(
"C:\\Users\\Public\\Pictures\\testLoginChrome.png"));

}

}

@Test(dependsOnMethods={"testChromeLogin"},description="After logging in to gmail", alwaysRun = true)

public void testInsideGmail() throws InterruptedException, IOException{

try {
//get the tab focused
Thread.sleep(1000L);
System.out.println("Page Title : " + _driver.getTitle());
Reporter.log("logged in success!!!");

} catch (Exception e) {

e.printStackTrace();
File sourceFile = ((TakesScreenshot) _driver)
.getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(sourceFile, new File(
"C:\\Users\\Public\\Pictures\\testInsideGmailChrome.png"));}}
```

Final word is that if you look at the code in Firefox and Chrome they are almost same. Therefore, I am going to skip putting rest of the code because they are identical almost in rest of the test cases.