

Test Automation :

SELENIUM

Automation ?

Automation is a process, which includes the development of scripts to run the test cases automatically (without manual intervention) and to log the results.

Why Automation ?

- To increase the Speed and also the accuracy of Testing
- It is time saving
- Regression testing.
- Performance testing
- Each future test cycle will take less time and require less human intervention

Why Automation ? (Contd...)

- To avoid the errors that human makes when they get tired after multiple repetitions
- The test program won't skip any tests by mistake.
- Essential for compatibility testing if required (OS/ Browser).
- Unattended Execution of Test scripts.

Uses of Automated Testing

- Load/stress tests - Very difficult to have very large numbers of human testers simultaneously accessing a system.
- Regression test suites - Tests maintained from previous releases; run to check that changes haven't caused faults.
- Sanity tests - Run after every new system build to check for obvious problems.
- Stability tests - Run the system for 24 hours to see that it can stay up.

What to Automate ?

- Functionality Testing
- Regression Testing
- Performance Testing

You can automate--

- When there are multiple releases of application.
- Only when the application under manual test is stable.
- On tests that need to be run for every build of the application.

When to Automate?

- ✓ Tests that need to be run for every build of the application (*sanity check, regression test*)
- ✓ Tests that use multiple data values for the same actions (*data-driven tests*)
- ✓ Tests that require detailed information from application internals (e.g., SQL, GUI attributes)
- ✓ Stress/load testing



**More repetitive execution?
Better candidate for automation.**

When Not Automate?

- ☒ Usability testing
 - "How easy is the application to use?"
- ☒ One-time testing
- ☒ "ASAP" testing
 - "We need to test NOW!"
- ☒ Ad hoc/random testing
 - based on intuition and knowledge of application
- ☒ Tests without predictable results



Improvisation required?
Poor candidate for automation.

When NOT to Automate?

- Automation cannot be used for a project that has very few releases.
- Automation testing cannot be used in usability type of testing.
- It cannot be used when the functionality of the application changes frequently.
- Test with unknown results cannot be automated.
- When the project doesn't have enough time

Tool Evaluation Criteria

- Ease of Use - easy to understand, easy to maintain & easy to install
- Platform support
- Defect Tracking
- Tool functionality
- Reporting capability
- Scripting language
- Support on Third party controls
- Add-on compatibility features
- Pricing

Automation Steps

1. Load libraries
2. Identify Objects
3. Create Script
4. Run Script
5. Synchronization
6. Verification
7. Data Driven Testing
8. Output values
9. Handling Exceptions
10. Run Test Suite
11. Defect Management
12. Regression Testing



SELENIUM

What is Selenium?

- Selenium is a robust set of tools that supports rapid development of test automation for web-based applications.
- It also helps to test the mobile application which has webforms.
- Selenium provides a rich set of testing functions specifically geared to the needs of testing of a web application.
- Selenium operations are highly flexible, allowing many options for locating UI elements and comparing expected test results against actual application behavior

Selenium Features

- Supports Cross Browser Testing. The Selenium tests can be run on multiple browsers.
- Allows scripting in several languages like Java, C#, Ruby, PHP and Python

History of Selenium

- Selenium is one of the most famous open source automation tool in the world.
- Selenium was originally developed by Jason Huggins in 2004.
- Later on ThoughtWorks developers also joined him to develop it further.
- He developed a Javascript library that could drive interactions with the page, allowing him to automatically rerun tests against multiple browsers
- That library eventually became Selenium Core, which underlies all the functionality of Selenium Remote Control (RC) and Selenium IDE.
- In 2009, the merger of selenium and webdriver resulted into Selenium 2.0, i.e WebDriver

Selenium Advantages

- Open Source
- Big user Base
- Robust
- Supports variety of languages.
- Also supports webform based mobile applications

Selenium Disadvantages

- It only supports Web and limited mobile applications

Selenium components

1. Selenium IDE

2. Selenium RC (Remote Control)

3. Selenium Grid

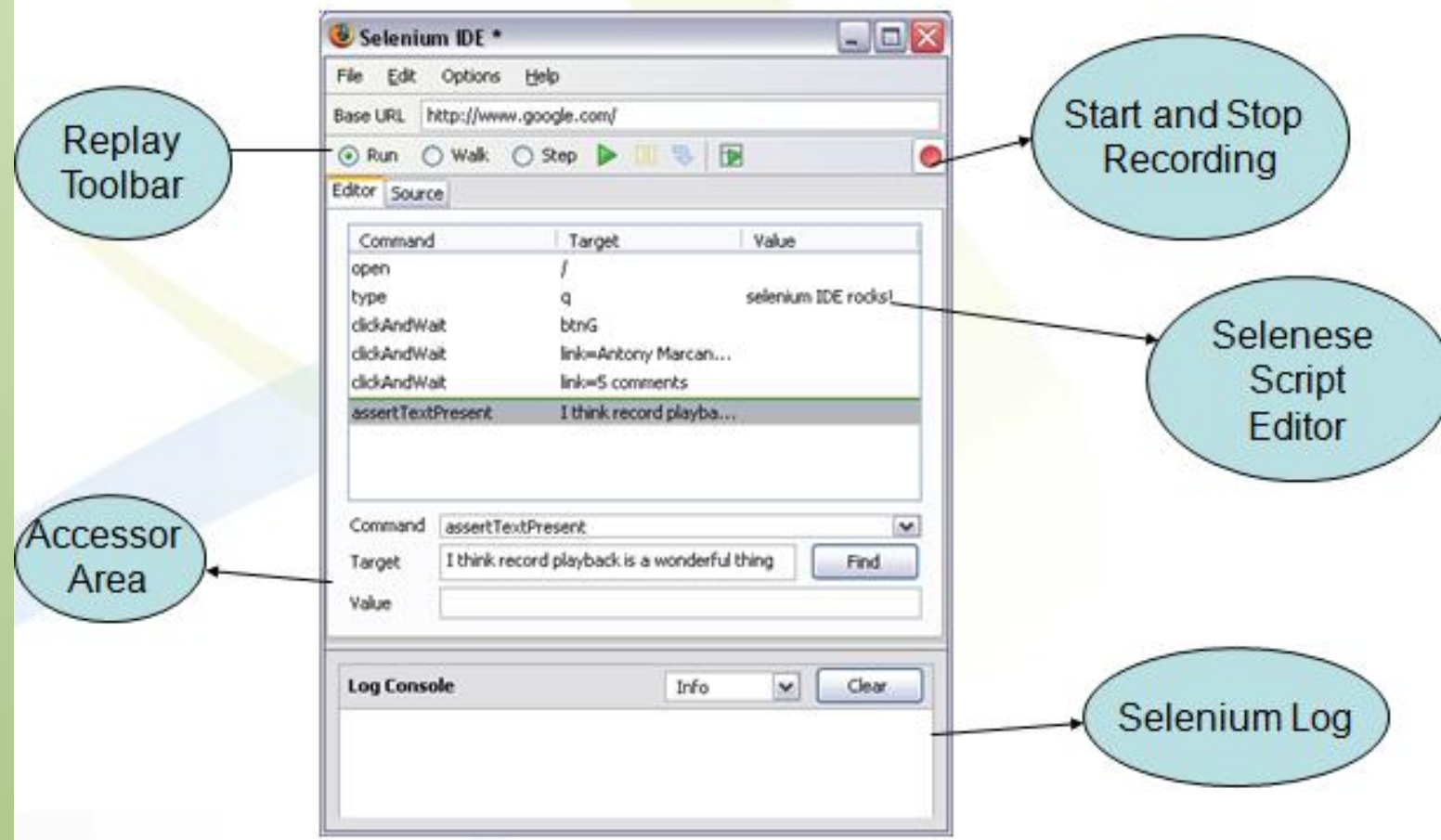
4. Selenium Webdriver:

Selenium IDE

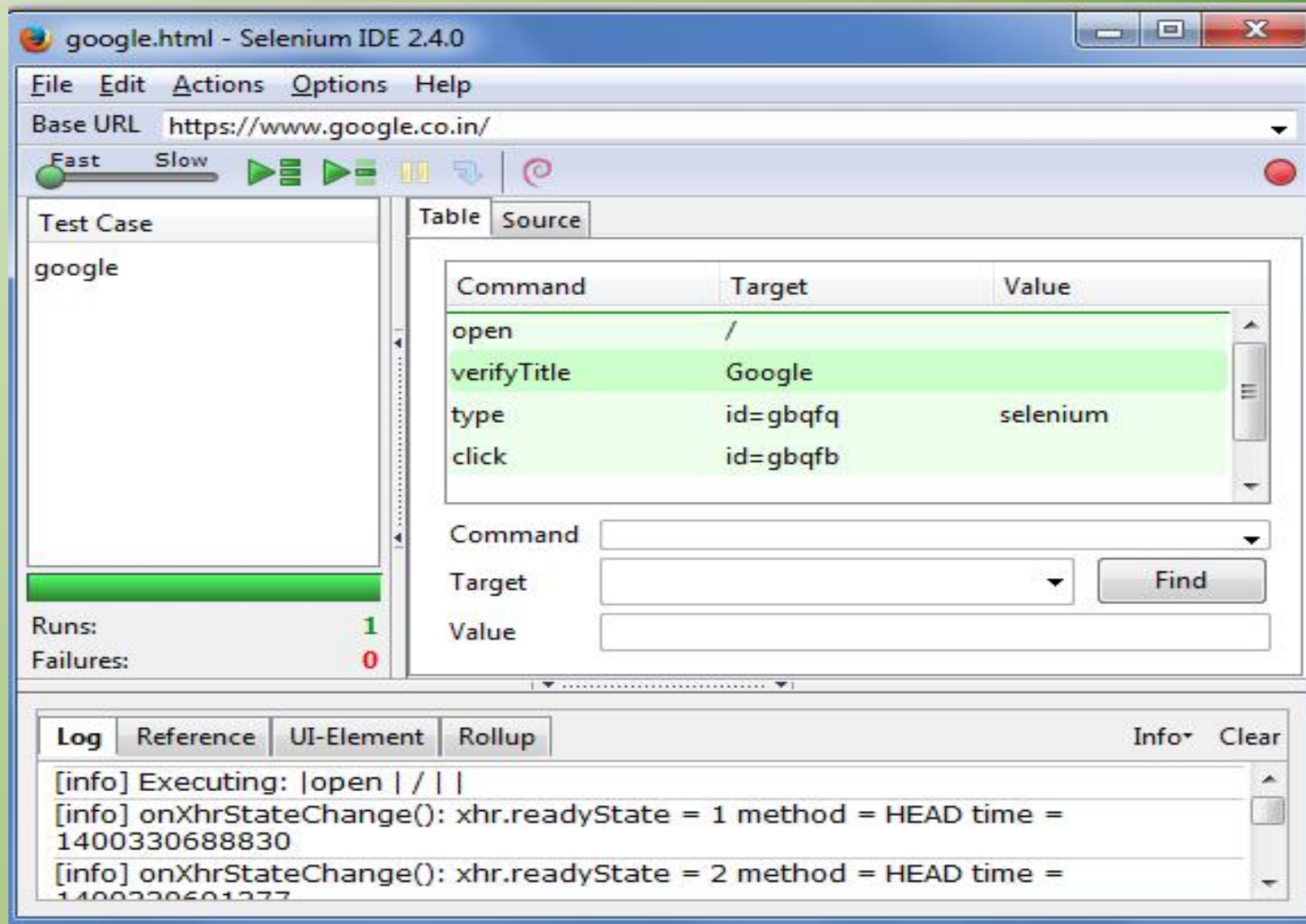
- Selenium IDE is an integrated development environment for Selenium tests.
- It is implemented as a Firefox extension, and allows you to record, edit, and replay the test in firefox
- Selenium IDE allows you to save tests as HTML, Java, Ruby ,Python or C# scripts
- Allows you to add selenese commands as and when required

Selenium IDE

Selenium IDE - UI



Selenium IDE



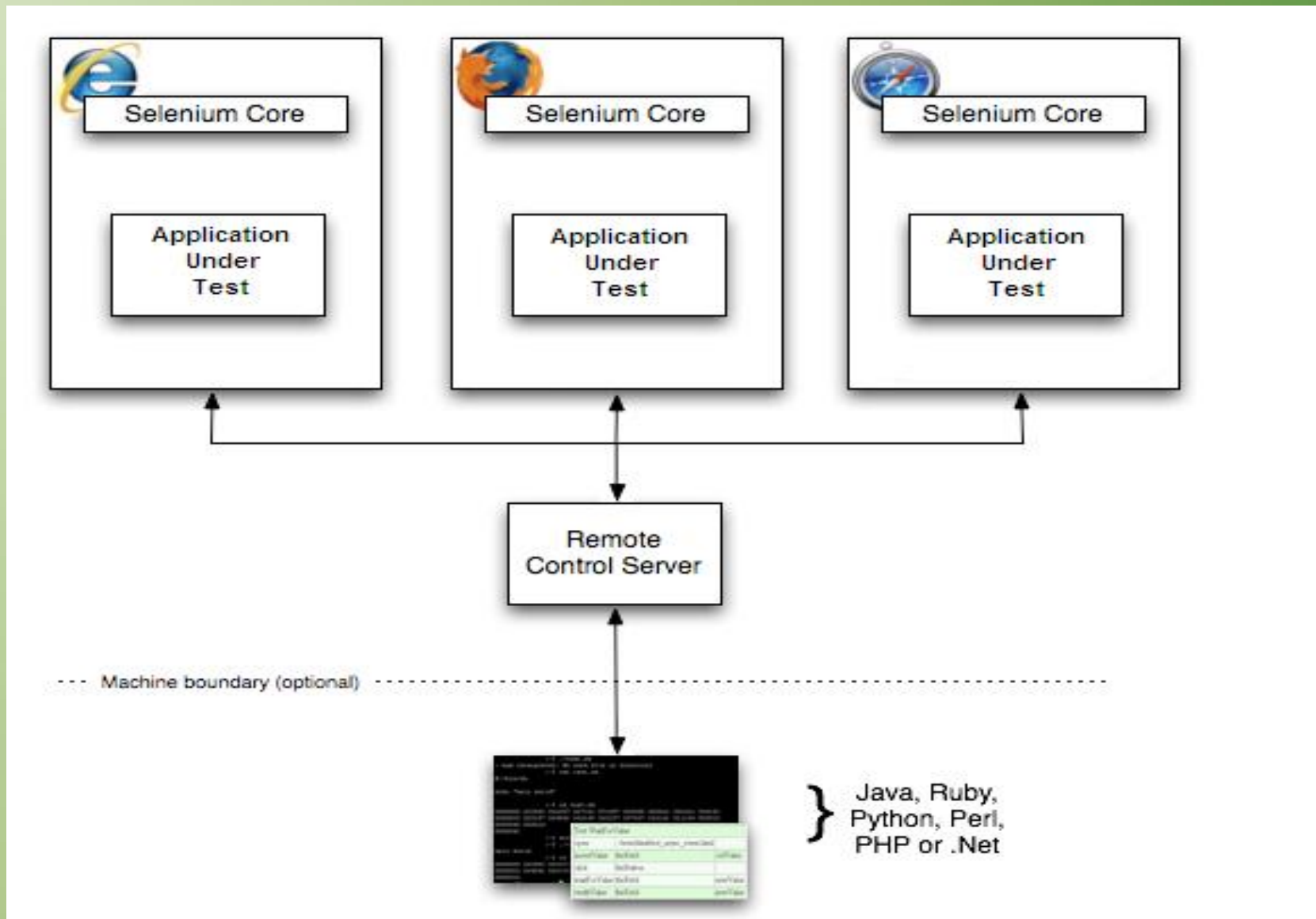
Limitations of Selenium IDE

- Can run the test only on Firefox
- No Programming logic (like loops, conditional statements) can be applied
- Selenium IDE can execute scripts created in Selenese only.
- It is difficult to use Selenium IDE for checking complex test cases involving dynamic contents

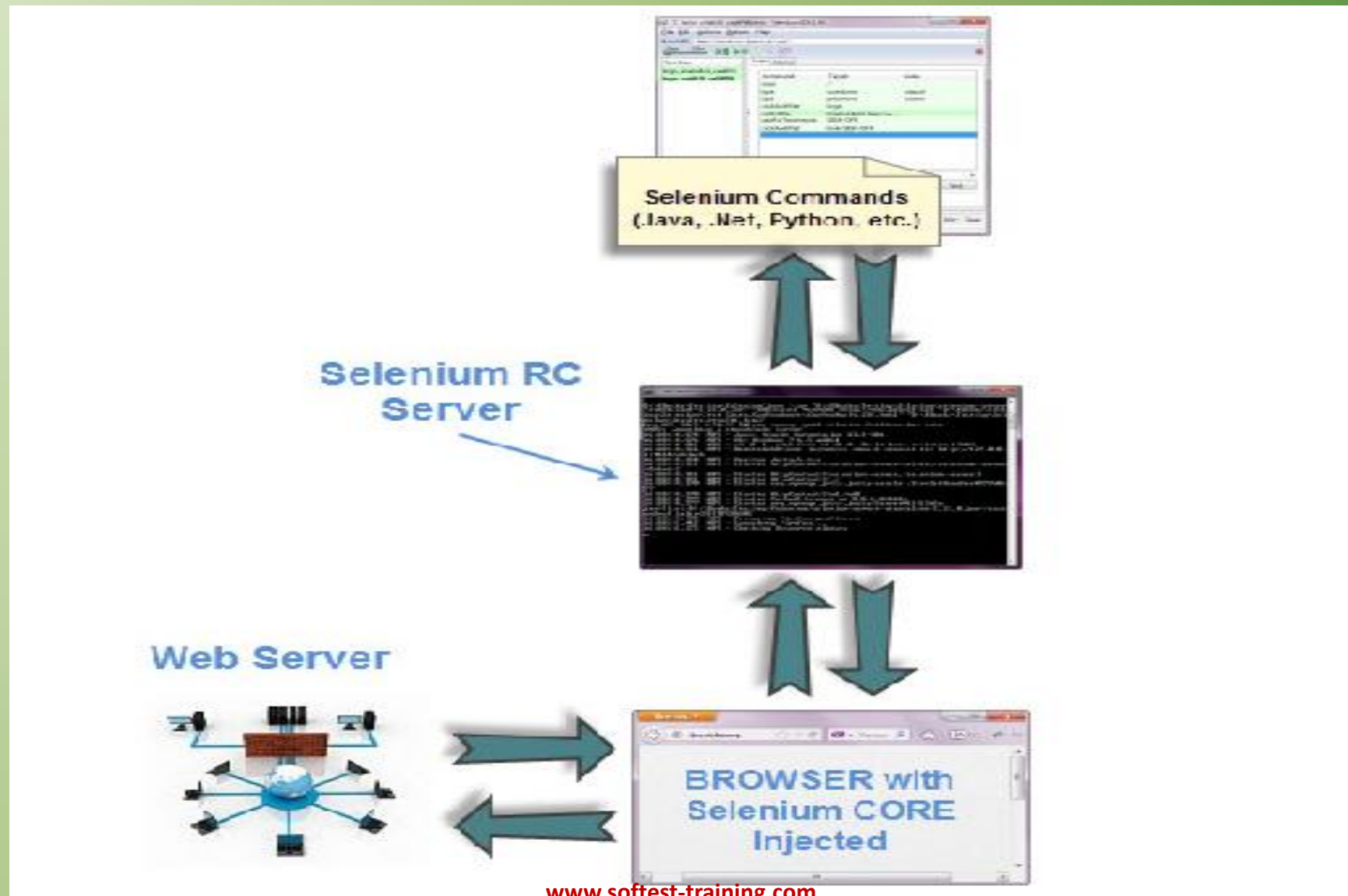
Selenium RC

- A solution to cross browser testing.
- A server, written in Java and so available on all the platforms.
- Acts as a proxy for web requests from them.
- Client libraries for many popular languages.
- Bundles Selenium Core and automatically loads into the browser
- Selenium RC automatically generates an HTML file of test results

Selenium RC Architecture



Selenium RC Architecture



Limitations of Selenium RC

- Selenium RC is slower since it uses a Javascript program called Selenium Core. This Selenium Core is the one that directly controls the browser.
- Selenium RC's API is though matured but contains redundancies and often confusing commands. At times, different browsers interprets each command in different ways too.
- Selenium RC does not support the headless HtmlUnit browser. It needs a real, visible browser to operate on.
- Selenium RC needs the help of the RC Server in order to communicate with the browser.

Selenium-Grid

- Selenium-Grid allows the Selenium-RC solution to scale for test suites or test suites to be run in multiple environments.
- With Selenium-Grid multiple instances of Selenium-RC can run on various operating system and browser configurations, each of these when launched register with a hub.
- When tests are sent to the hub they are then redirected to an available Selenium-RC, which will launch the browser and run the test.
- This allows to run tests in parallel, with the entire test suite theoretically taking only as long to run as the longest individual test

Selenium-Grid Architecture

Selenium Grid consists of 2 main parts:

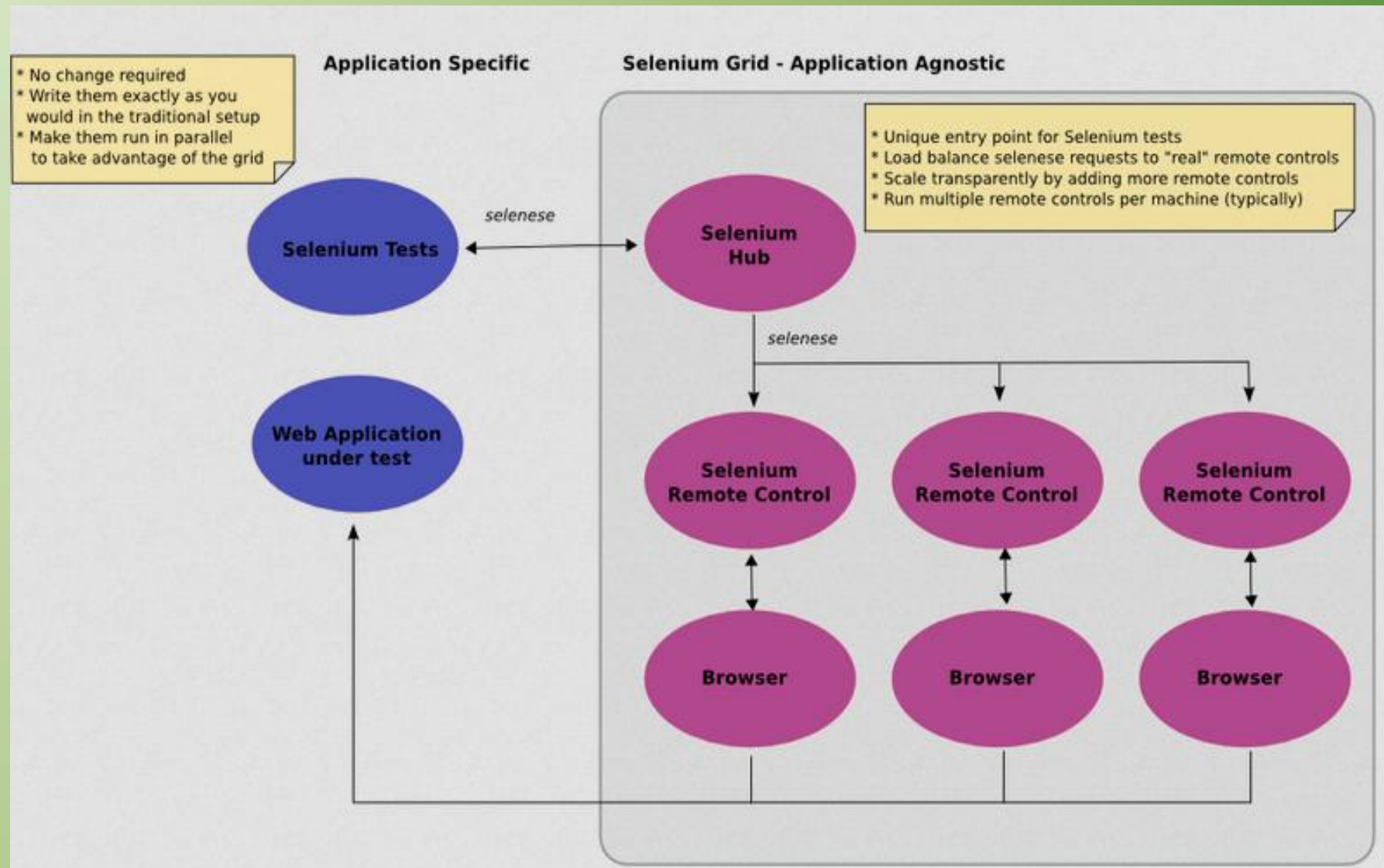
1. Selenium Hub:

- ✓ The main coordinator of all Selenium Remote Control.
- ✓ With appropriated mentioned environment, Selenium Hub will forward the request to specific environment in Selenium Remote Control.

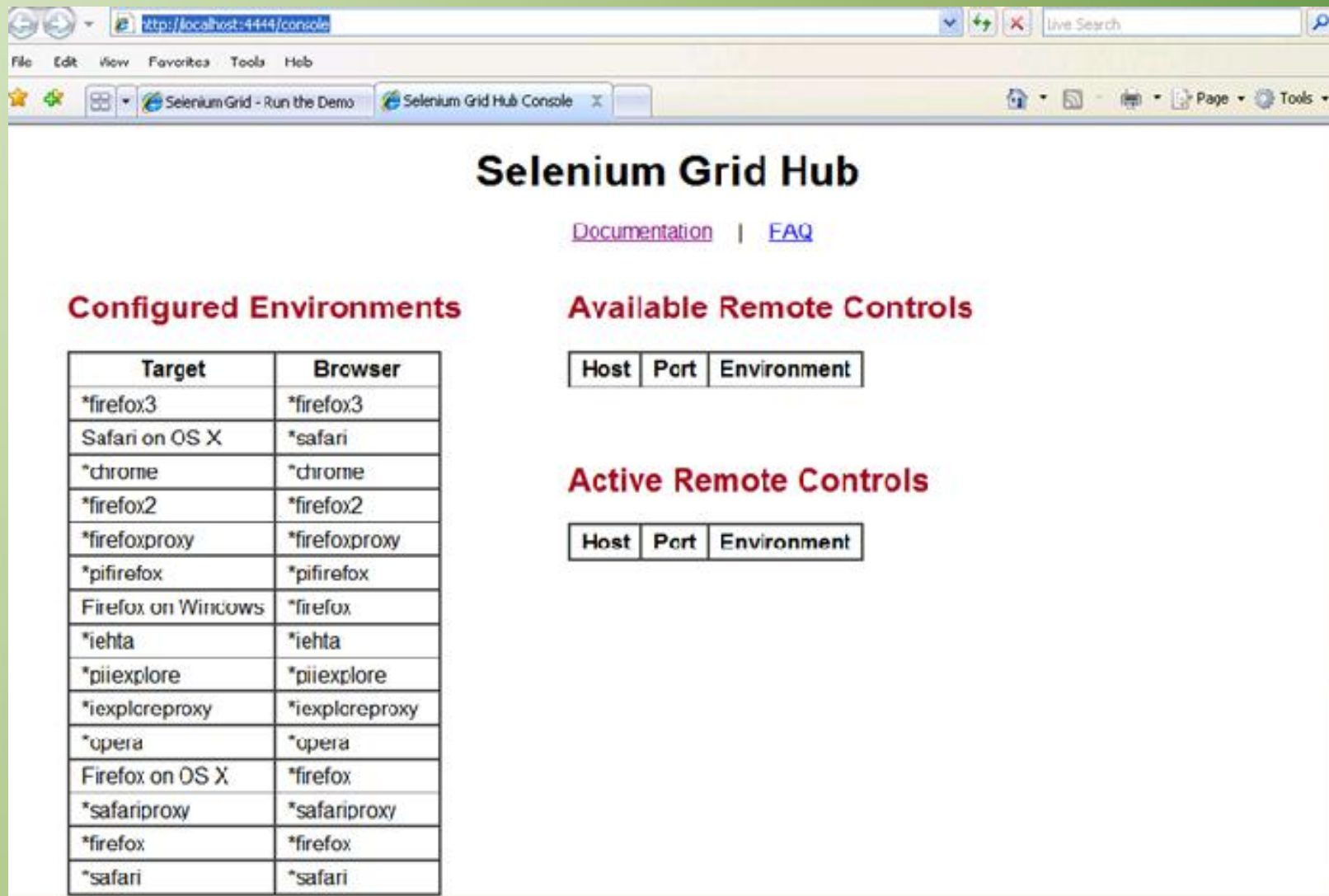
2. Selenium Remote Control (RC):

- ✓ The server that will invoke browser to be under test.
- ✓ It receive the command from Selenium Hub

Selenium-Grid Architecture



Selenium-Grid Architecture



The screenshot shows the Selenium Grid Hub console interface. The browser address bar indicates the URL is `http://localhost:4444/console`. The page title is "Selenium Grid Hub". Below the title, there are links for "Documentation" and "FAQ". The main content is divided into two sections: "Configured Environments" and "Available Remote Controls".

Configured Environments

Target	Browser
*firefox3	*firefox3
Safari on OS X	*safari
*chrome	*chrome
*firefox2	*firefox2
*firefoxproxy	*firefoxproxy
*pifirefox	*pifirefox
Firefox on Windows	*firefox
*iehta	*iehta
*piiexplore	*piiexplore
*iexplcreproxy	*iexplcreproxy
*opera	*opera
Firefox on OS X	*firefox
*safariproxy	*safariproxy
*firefox	*firefox
*safari	*safari

Available Remote Controls

Host	Port	Environment
------	------	-------------

Active Remote Controls

Host	Port	Environment
------	------	-------------

Selenium Webdriver

Selenium Webdriver

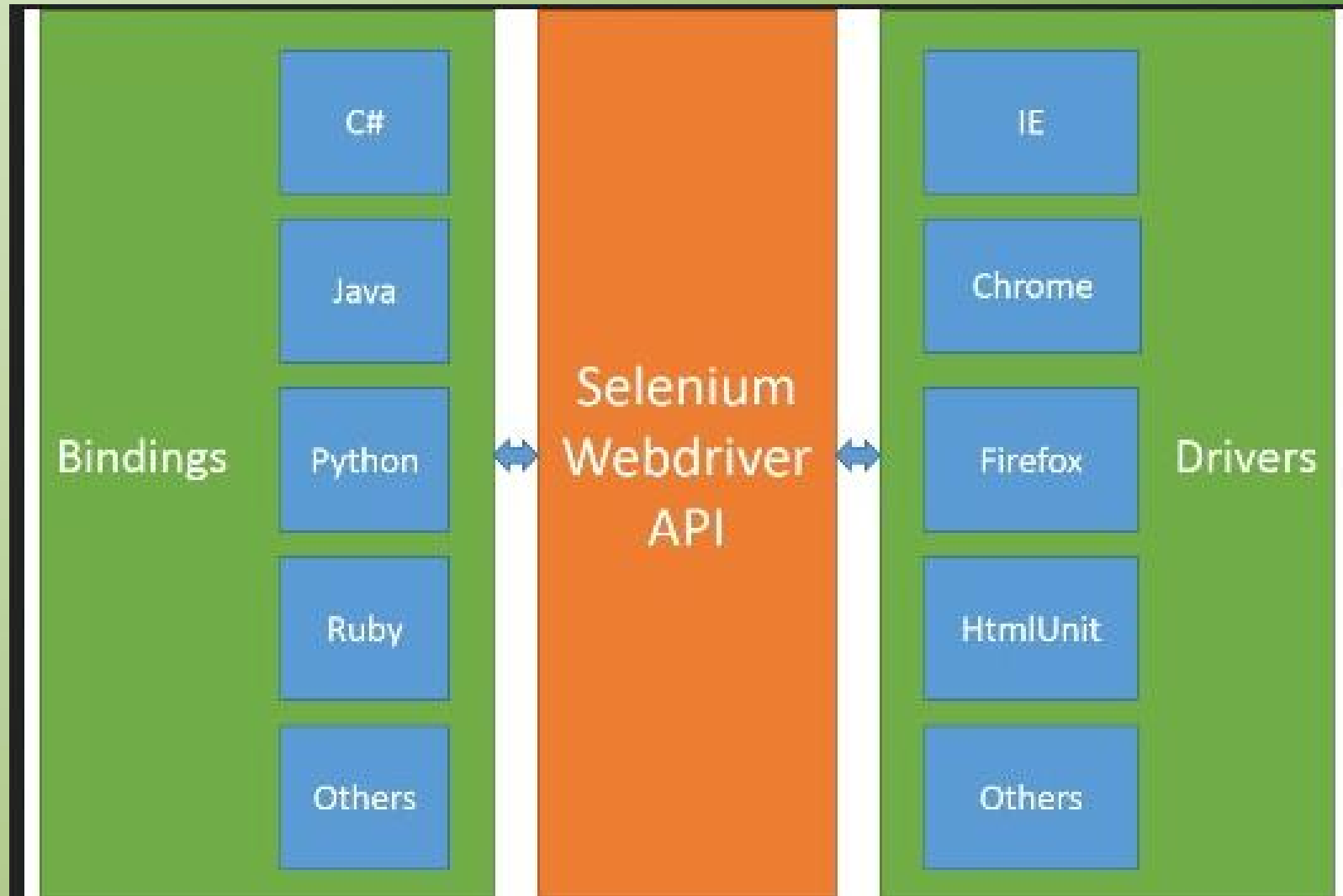
- WebDriver is designed in a simpler and more concise programming interface along with addressing some limitations in the Selenium-RC API.
- WebDriver is a compact Object Oriented API when compared to Selenium 1.0 (RC)
- WebDriver's API is more concise
- WebDriver supports the headless HtmlUnit browser
- WebDriver is faster than Selenium RC since it does not need any server.

Language support by WebDriver

Following programming languages are supported by WebDriver

- Java
- C#
- Python
- Ruby

Selenium Webdriver Architecture



Selenium Webdriver Architecture

Selenium webdriver architecture mainly divided into three parts

1. Language level bindings
2. Selenium Webdriver API
3. Drivers

Locators

Types of Locators in selenium webdriver

- `By.className`
- `By.cssSelector`
- `By.id`
- `By.linkText`
- `By.name`
- `By.partialLinkText`
- `By.tagName`
- `By.xpath`

Locator : By.className

1. By.className

See below example

Example:1

```
<td class=name> </td>
```

```
WebElement td=driver.findElement(By.className("name"));
```

Locator : By.cssSelector

2.By.cssSelector

CSS selector is the one the best ways to locate some complex elements in the page.

See below some examples for easy understanding

Example:1

```
<td class=name> </td>
```

```
driver.findElement(By.cssSelector("td.name"));    In css selector we can denote class name with dot (.)  
          (or)
```

```
driver.findElement(By.cssSelector("[class=name]"))  We can specify the attribute name and its value.
```

Example:2

```
<input id=create>
```

```
driver.findElement(By.cssSelector("#create")).sendKeys("test");  shortcut for denoting id is #  
          ( or )
```

```
driver.findElement(By.cssSelector("[id=create]")).sendKeys("test")
```

Example:3

```
<td value=dynamicValue_13232><td>
```

```
driver.findElement(By.cssSelector("[value*=dynamicValue]"))    * is for checking contained value  
(here value contains dynamicValue)
```

Example:4

```
<div value=testing name=dynamic_2307></div>
```

```
driver.findElement(By.cssSelector("[value=testing][name*=dynamic]"));
```

If you want to include more attribute values in locator criteria use css locator as above.

Locator : By.id

3. By.id

See below example

Example:1

```
<td id=newRecord> </td>
```

```
WebElement td=driver.findElement(By.id("newRecord"));
```

here we can specify the id attribute value directly.

Locator : linkText

4. By.linkText

See below example

Example:1

```
<a onclick=gotonext()>Setup </a>
```

```
WebElement link=driver.findElement(By.linkText("Setup"));
```

This is the best locator for locating links (anchor tags) in your web page.

Locator : partialLinkText

5. By.partialLinkText

See below example

Example:1

```
<a onclick=gotonext()>very long link text </a>
```

```
WebElement link=driver.findElement(By.partialLinkText("very"));
```

(or)

```
WebElement link=driver.findElement(By.partialLinkText("long link"));
```

This is the locator for locating links (anchor tags) using partial text it contains .

Locator : name

6. By.name

See below example

Example:1

```
<td name=WebDriver> </td>
```

```
WebElement td=driver.findElement(By.name("WebDriver"));
```

Locator : tagName

7. By.tagName

See below example

Example:1

```
<td class=name> </td>
```

```
WebElement td=driver.findElement(By.tagName("td"));
```

If you want to get the entire text of web page use below logic.

```
driver.findElement(By.tagName("body")).getText();
```

Locator : xpath

Finding elements with XPATH

- Absolute path

```
1 | WebElement userName =  
2 | driver.findElement(By.xpath("html/body/div/form/input"));
```

- Relative path

```
1 | WebElement email = driver.findElement(By.xpath("//input"));
```

Finding elements using index

```
1 | WebElement email =  
2 | driver.findElement(By.xpath("//input[3]"));
```

Finding elements using attributes values with XPath

```
1 | WebElement logo =  
2 | driver.findElement(By.xpath("img[@alt='logo']"));
```

XPATH

```
1 | starts-with()  
2 |  
3 | input[starts-with(@id,'input')]
```

Starting with:

```
1 | ends-with()  
2 |  
3 | input[ends-with(@id,'_field')]
```

Ending with:

```
1 | contains()  
2 |  
3 | input[contains(@id,'field')]
```

Locator : Preference Order

1. Id
2. Name
3. ClassName
4. Xpath
5. CSS
6. linkText
7. partialLinkText
8. tagName

WAIT

Different wait methods

1. Thread.Sleep()
2. implicitlyWait()
3. WebDriverWait (*Explicit Wait*)

Thread.sleep()

Not recommended

- You can never predict the exact wait time
- It increases overhead.
- It may not work with all computers

implicitlyWait()

- Tells WebDriver to wait for an element if they are not immediately available. So, WebDriver does not throw **NoSuchElementException** immediately.
- The default setting is 0. Once set, the implicit wait is set for the life of the WebDriver object instance.

```
driver.manage().timeouts().implicitlywait(15, TimeUnit.SECONDS);
```

```
WebDriver driver = new FirefoxDriver();  
driver.manage().timeouts().implicitlywait(10, TimeUnit.SECONDS);  
driver.get("http://somedomain/url_that_delays_loading");  
WebElement myDynamicElement = driver.findElement(By.id("myDynamicElement"));
```

Explicit Wait

- An explicit wait is code you define to wait for a certain condition to occur before proceeding further in the code.

```
WebDriver driver = new FirefoxDriver();  
driver.get("http://somedomain/url_that_delays_loading");  
WebElement myDynamicElement = (new WebDriverWait(driver, 10))  
.until(ExpectedConditions.presenceOfElementLocated(By.id("myDynamicElement")));
```

WebdriverWait

- Checks for the given condition every 500 milliseconds until it returns successfully or timeout.
- Another approach is to use ExpectedCondition and WebDriverWait strategy.
- The code below waits for 10 seconds or till the element is available, whichever is the earliest.

```
WebDriver driver = new FirefoxDriver();  
driver.get("http://google.co.in");  
WebDriverWait myDElement = new WebDriverWait(driver, 10)  
  
.until(ExpectedConditions.presenceOfElementLocated(By.xpath("xxxxx")));
```

Assert

Assert

The common types of assertions are:

- Is the page title as expected
- Validations against an element on the page

Assert : Page Title

You can get the current page title simply by calling getTitle() on the WebDriver instance. Here is how a simple test would look like:

```
1 @Test
2 public void pageTitle() {
3     driver.get("http://referencewebapp.qaautomation.net/");
4     String pageTitle = driver.getTitle();
5     assertEquals("Current page title", "Reference Web App - QA Automation", pageTitle);
6 }
```


Assert : Validations against an element

- Text within an element

```
1 driver.get("http://referencewebapp.qaautomation.net/");  
2 WebElement header = driver.findElement(By.id("header"));  
3 assertEquals("Header text", "Reference Web App", header.getText());
```

Assert : Attributes of the element

```
1 driver.get("http://referencewebapp.qaautomation.net/");  
2 WebElement submit = driver.findElement(By.name("submit"));  
3 assertEquals("Submit button value", "Login", submit.getAttribute("value"));
```

Cross Browser

Test Execution on different internet browsers

- AndroidWebDriver
- ChromeDriver
- EventFiringWebDriver
- FirefoxDriver
- HtmlUnitDriver
- InternetExplorerDriver
- IPHONEDriver
- IPHONESimulatorDriver
- RemoteWebDriver
- SafariDriver i.

Chrome Driver

- **Step 1:** Download the Chrome Webdriver
- **Step 2:** Define the ChromeDriver Path:
 - Either you set the path inside system PATH or you can define it inside the code. OR Use following code.

```
System.setProperty("webdriver.chrome.driver",  
    "/path/to/chromedriver");
```

- **Step 3:** Creating an Object of Chrome Driver:

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

Internet Explorer Driver

- **Step 1:** Download the Internet Explorer Webdriver
- **Step 2:** Define the Internet Explorer Driver Path:
 - Either you set the path inside system PATH or you can define it inside the code. OR Use following code.

```
System.setProperty("webdriver.ie.driver",  
"/path/to/IEDriver");
```

- **Step 3:** Creating an Object of Internet Explorer Driver:

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

Firefox Driver

- **Step 1:** Download the GeckoDriver
- **Step 2:** Define the GeckoDriver Path:
 - Either you set the path inside system PATH or you can define it inside the code. OR Use following code.

```
System.setProperty("webdriver.gecko.driver",  
"/path/to/GeckoDriver");
```

- **Step 3:** Creating an Object of Internet Explorer Driver:

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

HtmlUnit Driver

- HtmlUnit Driver is the fastest and most lightweight implementation of WebDriver
- A pure Java solution and so it is platform independent.
- Supports Javascript

```
HtmlUnitDriver myDriver = new HtmlUnitDriver();
```


Advanced Selenium

Handling JavaScript PopUps

Usually, a web application can generate following types of popup windows:

JavaScript PopUps

Browser PopUps

Native OS PopUps



JavaScript popup windows are generated by the web application code.
Hence they can be controlled by the browser.



Native Popups like the File Upload / Download windows call the operating system's native interface. Hence once they are opened, the browser has little control over them.

Handling JavaScript PopUps (Alerts)

Types of Alerts:

1. Confirmation Box
2. Prompt PopUp
3. Authentication box

Handling JavaScript PopUps (Alerts)

WebDriver Alert Interface

1. Void dismiss();
2. Void accept();
3. String getText();
4. Void sendKeys(String ketsToSend);

Selenium provides an API to handle JavaScript PopUps.

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

The Alert API allows various operations on the Alert object:

```
accept(), dismiss(), getText(), sendKeys()
```

Handling window popups

As Selenium does not provide support to these kind of Popups, we need to explore other options. There are several ways to handle such popups like:

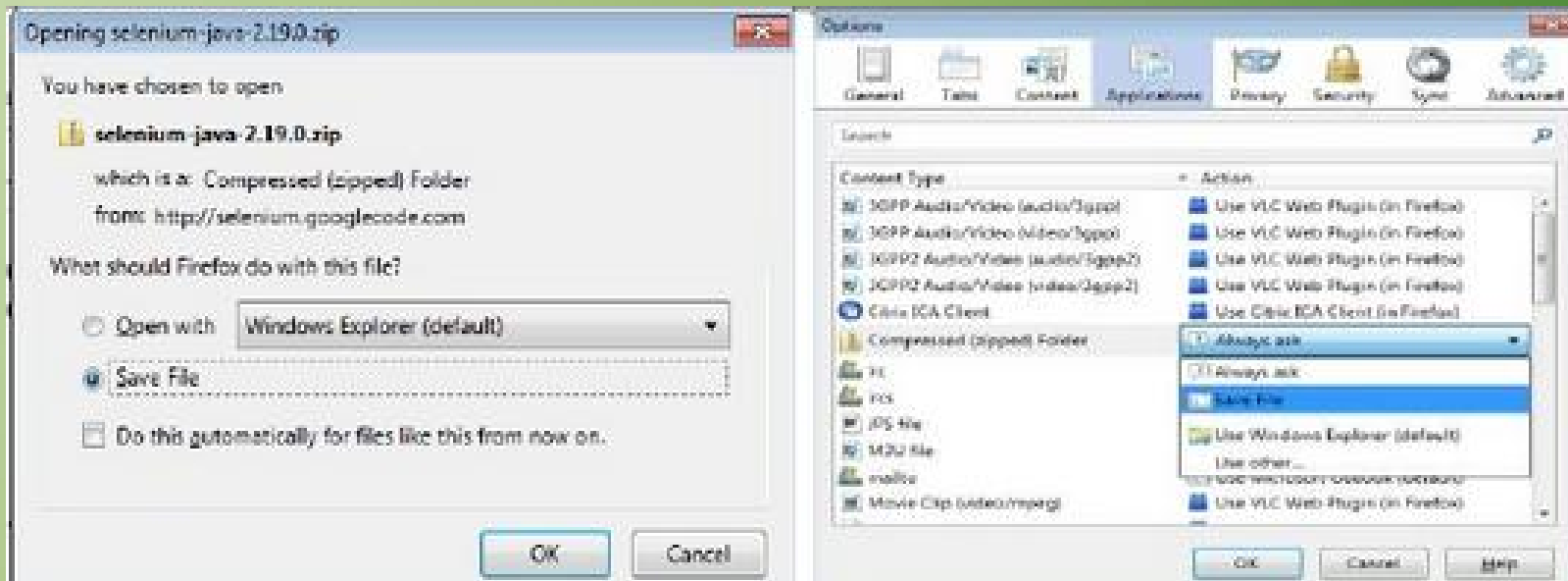
Third party tools to handle window pop-ups

- AutoIT
- Robot Class (Java.awt.Robot Toolkit)

Handling browser and window popups

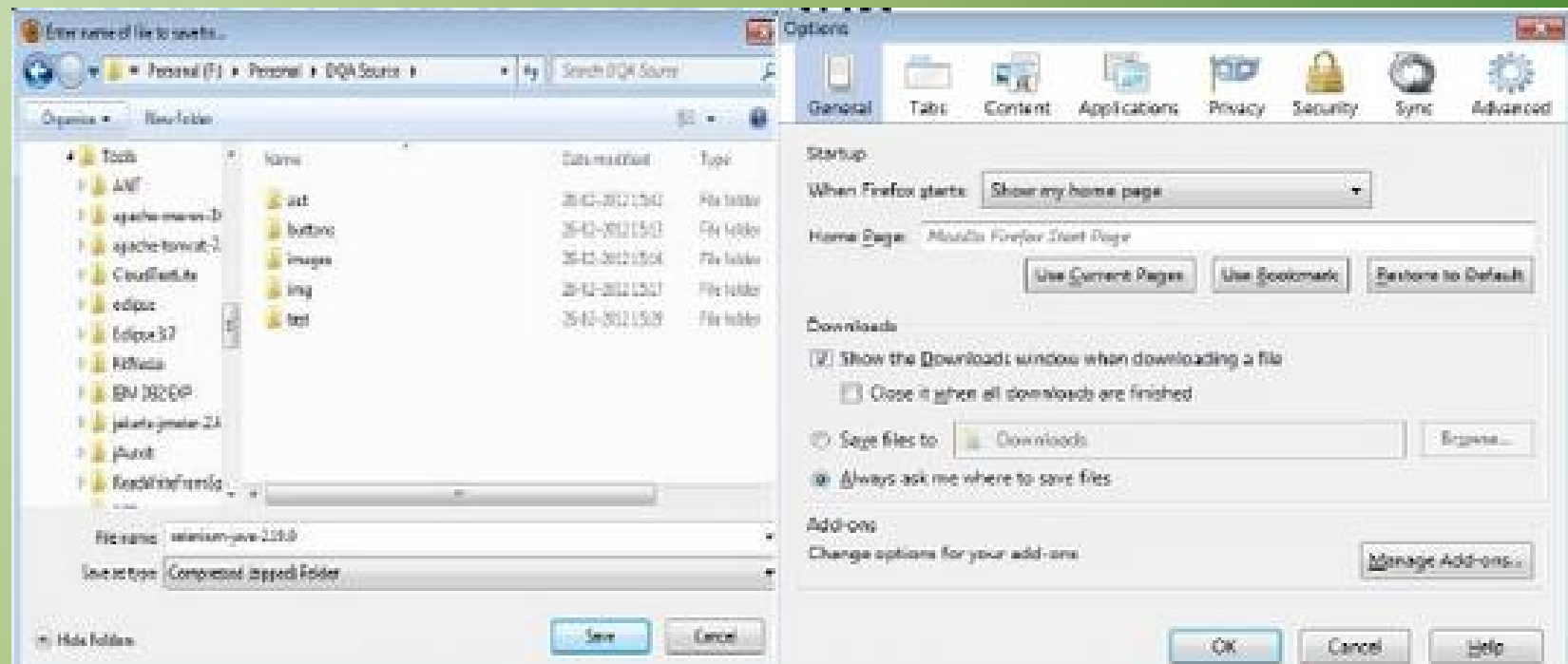
- A useful way to handle popups is to set the browser default to disable popups wherever possible.

Eg; PopUp asking where a file should be saved



Handling browser and window popups

- When a file is to be downloaded, the browser will ask you where the file has to be saved. This option can be set to a default.



Java AWT Robot Class

This class can generate native input events to the underlying OS using the Keyboard and Mouse Interfaces

Methods in this class can be effectively used to do the minimal interaction with native popups occurring in Web Applications.

Some of the useful API provided are:

- *keyPress()*;
- *mousePress()*;
- *Mousemove()*;

Although the Robot Class can be used to interact with PopUps, there are some limitations like:

- All interactions happen using either Keyboard events or screen coordinates.
- Keyboard events handle 1 character at a time (no Strings)
- Cannot capture object properties using this class.

Java AWT Robot Class

```
Robot rb=new Robot();  
Rb.keyPress(keyEvent.VK_S);  
Rb.keyRelease(keyEvent.VK_S);
```

```
Rb.keyPress(keyEvent.VK_M);  
Rb. keyRelease(keyEvent.VK_M);
```

```
Rb.keyPress(keyEvent.VK_I);  
Rb. keyRelease(keyEvent.VK_I);
```

```
Rb.keyPress(keyEvent.VK_T);  
Rb. keyRelease(keyEvent.VK_T);
```

```
Rb.keyPress(keyEvent.VK_A);  
Rb. keyRelease(keyEvent.VK_A);
```

AutoIT

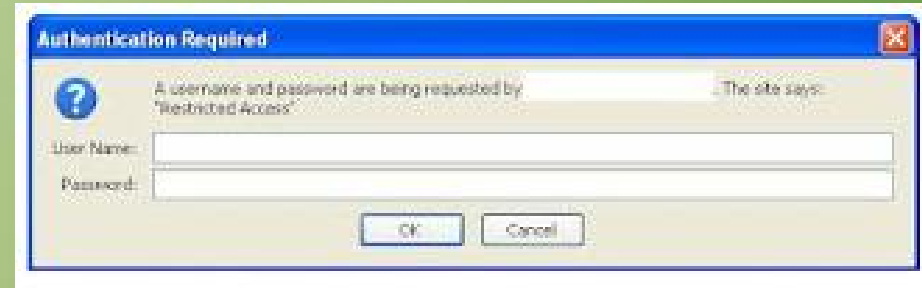
- AutoIT is a tool that can automate the window GUI
- It uses a combination of simulated keystrokes, mouse movement and window/control manipulation in order to automate tasks .
- AutoIT generates an executable file that can be called from selenium script

AutoIT Code Example:

```
WinWaitActive("Choose File to Upload")  
Send("D:\test.jpeg")  
Send("{ENTER}")
```

AutoIT

Ex: Using AutoIT for Handling the Authentication PopUp



1. Write the below code and Save it as 'authenticate.au3' (au3 is the AutoIT format)

```
WinWaitActive("Authentication Required")
```

```
Send("Username")
```

```
Send("{TAB}")
```

```
Send("Password")
```

```
Send("{ENTER}")
```

2. Compile the script to convert into an executable file

3. Add the below line in selenium code

```
Runtime.getRuntime().exec("C://Path//authenticate.exe");
```

FrameWork in Selenium with Java

- Junit
- TestNG

Automation Framework

- Linear Scripting
- Modular
- Library Architecture
- Data Driven
- KeyWord
- Hybrid

Behavioural Driven Development

- Cucumber
- JBehave