# Selenium Introduction

Selenium – Web and mobile applications

For desktop – SIKULI & AutoIT (add-on automation tool)

1. Selenium IDE

Record and playback

Add-on for firefox (only)

It can be recorded only in firefox, but can be played back in other browsers, through RC and WebDriver.

If app only compatible with IDE or Chrome no other options.

1. Selenium RC

Remote control

Act as Server/API

1. WebDriver

To overcome the disadvantages of RC

1. Grid

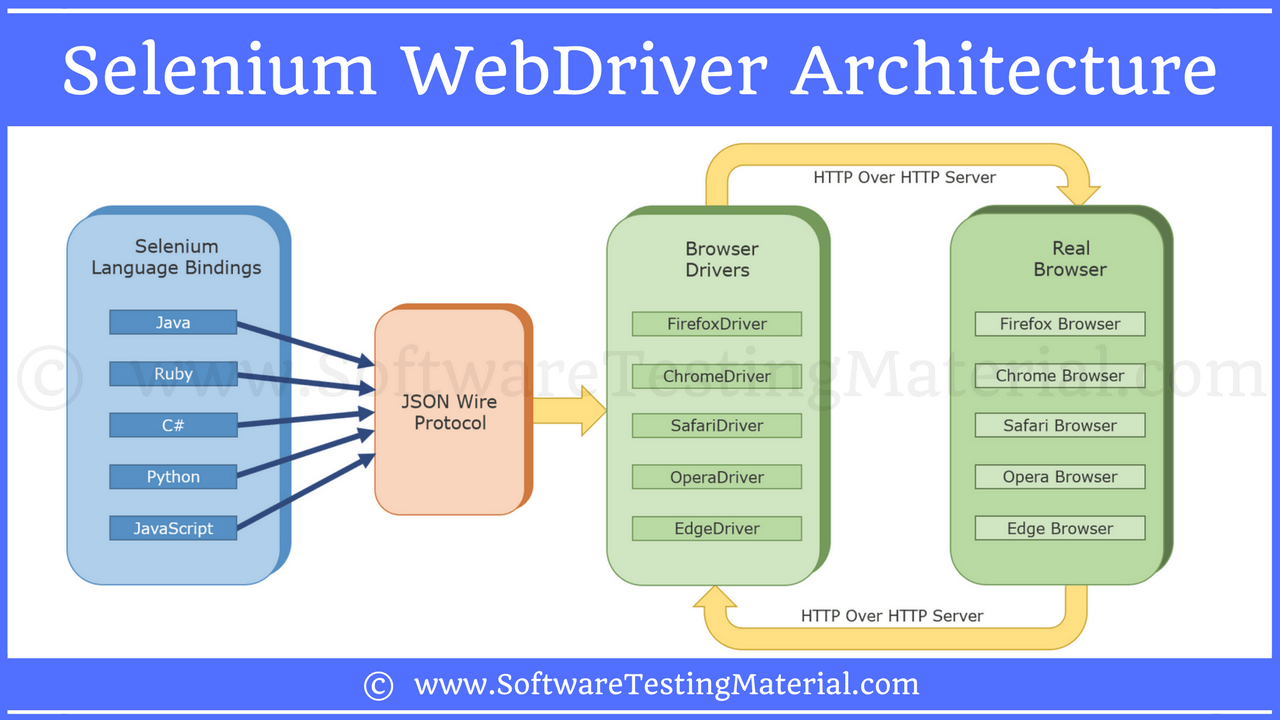
For parallel execution

100 Test cases (Simultaneously in Chrome, Safari, Firefox, IE, Opera)

**Selenium configuration**

1. Eclipse
2. JDK/JRE 8
3. Selenium standalone server 2.44.0 jar

# WebDriver Architecture



## **Selenium Client Libraries/Language Bindings:**

Language bindings allow Selenium to support multiple languages like Java, Ruby, Python, etc.

## **JSON Wire Protocol over HTTP Client:**

JSON (JavaScript Object Notation) is a **lightweight data-interchange format**.

It is used to transfer data between a server and a client on the web over HTTP server.

## **Browser Drivers:**

Each browser contains separate browser driver. When a browser driver is received any command then that command will be executed on the respective browser and the response will go back in the form of HTTP response.

## **Browsers:**

Selenium supports multiple browsers such as Firefox, Chrome, IE, Safari etc.

In real time, you write a code in your UI (say Eclipse IDE) using any one of the supported Selenium client libraries (say Java).

### Function

Once you click on Run, every statement in your script will be converted as a URL with the help of JSON Wire Protocol over HTTP. The URL’s will be passed to the Browser Drivers.

Every Browser Driver uses a HTTP server to receive HTTP requests.  Once the URL reaches the Browser Driver, then the Browser Driver will pass that request to the real browser over HTTP. Then the commands in your selenium script will be executed on the browser.

If the request is *POST* request then there will be an action on browser

If the request is a *GET* request then the corresponding response will be generated at the browser end and it will be sent over HTTP to the browser driver and the Browser Driver over JSON Wire Protocol and sends it to the UI (Eclipse IDE).

This is all about Selenium WebDriver Architecture.

# Selenium Locators

Locators provide a way to access the HTML elements from a web page.

1. Locating by id

Driver.findElement(By.id(“id name”))

1. Locating by name

Driver.findElement(By.name(“name”))

1. Locating by tagname

Select x=new Select(By.tagName(“tagname”))

1. Locating by LinkText

Driver.findElement(By.linkText(“name”))

1. Locating by Partial LinkText

Driver.findElement(By.partialLinkText(“name”))

1. Locating by class name

Driver.findElement(By.className(“name”))

1. Locating by CSS Locators

Driver.findElement(By.cssSelector(“name”))

Used to find webelements inside frames

1. Locating by Xpath

Driver.findElement(By.xpath(“name”))

# WebDriver Commands

driver.get(“url”); – Load page in the current window

.getTitle(); – Fetches the current page title

.getCurrentUrl(); – current page URL

.getPageSource();S – Source code of the page

.close(); – To close the current window

.quit(); – To close all the windows

.getClass() – Run time class of the class object

.getText() – get inner text of the specified web element

.getAttribute() - retrieve the value of the specified attribute

.getWindowHandle(); - get current window’s unique id (String)

.getWindowHandles(); - Get all windows’ unique IDs (Store in a Set as string)

.isEnabled() - Check Whether the Element is Enabled Or Disabled

.navigate().to("URL"); - Launch URL

# Select Class/Drop down handling

1. Single select dropdown
2. Multi select dropdown

Select select = new Select(driver.findElement(By.id("SelectID\_One")));

select.selectByValue("greenvalue");

select.selectByVisibleText("Red");

select.selectByIndex(2);

# Action Class

To perform mouse actions

Actions actions=new Actions(driver);

actions.moveToElement(Element).build().perform();

click (): Simply click on element

doubleClick (): Double clicks onElement

contextClick(): Performs a right click on an element

clickAndHold(): Clicks at the present mouse location (without releasing)

dragAndDrop(source, target): Clicks in source location and moves to the location of the target element

moveToElement(toElement): It shifts the mouse to the center of the element

release(): Releases the depressed left mouse button at the existing mouse location

# JavascriptExecutor (Scrolling)

Scenario 1: To scroll down the web page by pixel.

((JavascriptExecutor)driver).executeScript("window.scrollBy(0,1000)");

Scenario 2: To scroll down the web page at the bottom of the page.

((JavascriptExecutor)driver).executeScript("window.scrollTo(0,document.body.scrollHeight)");

Scenario 3: To scroll down the web page by the visibility of the element.

((JavascriptExecutor)driver).executeScript("arguments[0].scrollIntoView();", driver.findElement(By.linkText("VBScript")));

Scenario 4: Horizontal scroll on the web page.

((JavascriptExecutor)driver).executeScript("arguments[0].scrollIntoView();", driver.findElement(By.linkText("VBScript")));

# Screenshot

FileUtils.copyFile(((TakesScreenshot)driver).getScreenshotAs(OutputType.*FILE*), new File("C:/selenium/error.png"));

# Alert & Popup

1. driver.switchTo().alert().dismiss();
2. driver.switchTo().alert().accept();
3. driver.switchTo().alert().getText();
4. driver.switchTo().alert().sendKeys("Text");

# Excel Read/Write

Excel Read/Write

**Read**

FileInputStream fis=new FileInputStream(“path”);

XSSFWorkbook workbook=new XSSFWorkbook(fis);

XSSFSheet sheet=workbook.getSheetAt(0);

Row row=sheet.getRow(0);

Cell cell=row.getCell(0);

Cell.getCellValue();

**Write**

FileOuputStream fis=new FileOuputStream(“path”);

XSSFWorkbook workbook=new XSSFWorkbook(fis);

XSSFSheet sheet=workbook.createSheetAt(0);

Row row=sheet.createRow(0);

Cell cell=row.createCell(0);

Cell.setCellValue();

***For different form use this link***

<https://howtodoinjava.com/apache-commons/readingwriting-excel-files-in-java-poi-tutorial/>

Example read:

package com.howtodoinjava.demo.poi;

//import statements

public class ReadExcelDemo

{

    public static void main(String[] args)

    {

        try

        {

            FileInputStream file = new FileInputStream(newFile("howtodoinjava\_demo.xlsx"));

            //Create Workbook instance holding reference to .xlsx file

            XSSFWorkbook workbook = new XSSFWorkbook(file);

            //Get first/desired sheet from the workbook

            XSSFSheet sheet = workbook.getSheetAt(0);

            //Iterate through each rows one by one

            Iterator<Row> rowIterator = sheet.iterator();

            while (rowIterator.hasNext())

            {

                Row row = rowIterator.next();

                //For each row, iterate through all the columns

                Iterator<Cell> cellIterator = row.cellIterator();

                while (cellIterator.hasNext())

                {

                    Cell cell = cellIterator.next();

                    //Check the cell type and format accordingly

                    switch (cell.getCellType())

                    {

                        case Cell.CELL\_TYPE\_NUMERIC:

                            System.out.print(cell.getNumericCellValue() + "t");

                            break;

                        case Cell.CELL\_TYPE\_STRING:

                            System.out.print(cell.getStringCellValue() + "t");

                            break;

                    }

                }

                System.out.println("");

            }

            file.close();

        }

        catch (Exception e)

        {

            e.printStackTrace();

        }

    }

}

Output:

ID      NAME        LASTNAME

1.0     Amit        Shukla

2.0     Lokesh      Gupta

3.0     John        Adwards

4.0     Brian       Schultz

Example write:

static void basedOnValue(Sheet sheet)

{

    //Creating some random values

    sheet.createRow(0).createCell(0).setCellValue(84);

    sheet.createRow(1).createCell(0).setCellValue(74);

    sheet.createRow(2).createCell(0).setCellValue(50);

    sheet.createRow(3).createCell(0).setCellValue(51);

    sheet.createRow(4).createCell(0).setCellValue(49);

    sheet.createRow(5).createCell(0).setCellValue(41);

    SheetConditionalFormatting sheetCF = sheet.getSheetConditionalFormatting();

    //Condition 1: Cell Value Is   greater than  70   (Blue Fill)

    ConditionalFormattingRule rule1 = sheetCF.createConditionalFormattingRule(ComparisonOperator.GT, "70");

    PatternFormatting fill1 = rule1.createPatternFormatting();

    fill1.setFillBackgroundColor(IndexedColors.BLUE.index);

    fill1.setFillPattern(PatternFormatting.SOLID\_FOREGROUND);

    //Condition 2: Cell Value Is  less than      50   (Green Fill)

    ConditionalFormattingRule rule2 = sheetCF.createConditionalFormattingRule(ComparisonOperator.LT, "50");

    PatternFormatting fill2 = rule2.createPatternFormatting();

    fill2.setFillBackgroundColor(IndexedColors.GREEN.index);

    fill2.setFillPattern(PatternFormatting.SOLID\_FOREGROUND);

    CellRangeAddress[] regions = {

            CellRangeAddress.valueOf("A1:A6")

    };

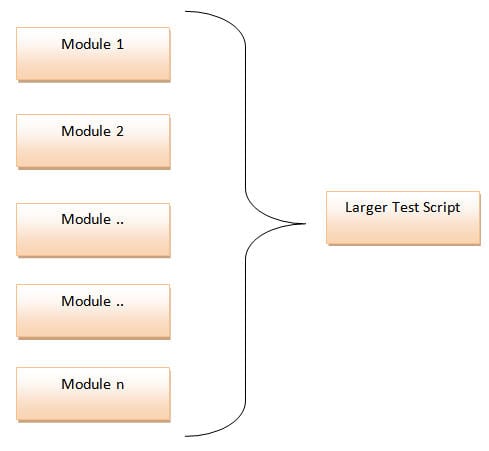
    sheetCF.addConditionalFormatting(regions, rule1, rule2);

}

# Frameworks

Let us discuss the few most popularly used Test Automation Frameworks:

1. Module Based Testing Framework
2. Library Architecture Testing Framework
3. Data Driven Testing Framework
4. Keyword Driven Testing Framework
5. Hybrid Testing Framework
6. Behavior Driven Development Framework
7. Modular driven framework



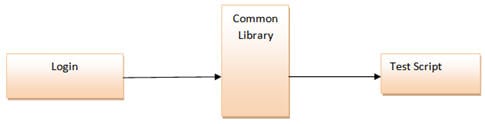
The framework divides the entire “Application Under Test” into a number of logical and isolated modules. For each module, we create a separate and independent test script. Thus, when these test scripts took together builds a larger test script representing more than one modules.

These modules are separated by an abstraction layer in such a way that the changes made in the sections of the application doesn’t yield affects on this module.

1. Library Architecture Testing Framework

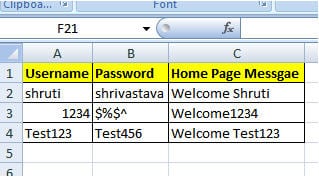
The basic fundamental behind the framework is to determine the common steps and group them into functions under a library and call those functions in the test scripts whenever required.

Example: The login steps can be combined into a function and kept into a library. Thus all the test scripts those require to login the application can call that function instead of writing the code all over again.

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/Test-Automation-Frameworks-4.jpg)

1. Data driven framework

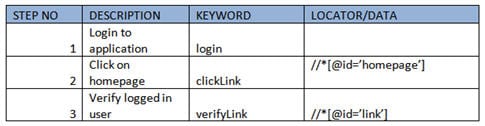
Data Driven Testing Framework helps the user segregate the test script logic and the test data from each other. It lets the user store the test data into an external database. The external databases can be property files, xml files, excel files, text files, CSV files, ODBC repositories etc. The data is conventionally stored in “Key-Value” pairs. Thus, the key can be used to access and populate the data within the test scripts.



Code to get data from excel in (TestUtil.util) → @Dataprovider (object array – two dimensional) → TestCase

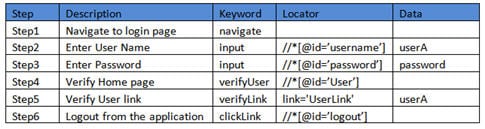
1. Keyword Driven Testing Framework

The Keyword driven testing framework is an extension to Data driven Testing Framework in a sense that it not only segregates the test data from the scripts, it also keeps the certain set of code belonging to the test script into an external data file.



1. Hybrid driven framework

As the name suggests, the Hybrid Testing Framework is a combination of more than one above mentioned frameworks. The best thing about such a setup is that it leverages the benefits of all kinds of associated frameworks.



1. Behavior Driven Development Framework

Though the above pictorial representation of a framework is self-explanatory we would still highlight a few points.

1. **Object Repository:** Object Repository acronym as OR is constituted of the set of locators types associated with web elements.
2. **Test Data:** The input data with which the scenario would be tested and it can be the expected values with which the actual results would be compared.
3. **Configuration File/Constants/ Environment Settings:** The file stores the information regarding the application URL, browser-specific information etc. It is generally the information that remains static throughout the framework.
4. **Generics/Program logics/Readers:** These are the classes that store the functions which can be commonly used across the entire framework.
5. **Build tools and Continuous Integration:** These are the tools that aids to the capabilities of the framework to generate test reports, email notifications and logging information.

# Window Handling

String mainWindow=driver.getWindowHandle();

Set<String> set =driver.getWindowHandles();

Iterator<String> itr= set.iterator();

while(itr.hasNext())

{

if(!mainWindow.equals(itr.next()))

{

driver.switchTo().window(itr.next());

driver.close();

}

driver.switchTo().window(mainWindow);

# WebTables

Public class dynamic\_table

{

PSVM (string [] args) throws interruptException

{

WebDriver driver = new firefoxDriver();

Driver.get(“url”);

WebElement dynamictable=driver.findElement(By.xpath(“path”));

List<WebElement> rows\_table=dynamictable.findElements (by.tagname(“tr”));

Int rows\_count=rows\_table.size();

For(int row=0; row)

# Synchronization

Two or more components involved to perform any action, we expect these components should be worked together with same pace

Exceptions are

* ElementNotVisibleException
* NoSuchElementException

In such scenarios, we use Synchronization/wait to overcome these kinds of exceptions.

Automation Testing is commonly comprising of following two segments:

1. Unconditional Synchronization – thread.sleep(times) – not advisable
2. Conditional Synchronization

**Conditional**

**Implicit wait**

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

Normally, it is not recommended to use implicit waits, when we can use explicit waits or fluent waits.

**Explicit wait**

Explicit Wait tells the WebDriver to Wait until the specified condition is met or maximum time elapses before throwing NoSuchElement (or) ElementNotVisible Exceptions.

We must first create instance for “WebDriverWait” class.

WebDriverWait wait = new WebDriverWait(driver, 10);

wait.until(ExpectedConditions.VisibilityOfElementLocated(By.id("someid"))).click();

We would normally use explicit wait if an element takes a long time to load. We also used explicit wait to check CSS property of an element (presence, clickability. etc) which can change in Ajax applications.

Following are the ExpectedConditions:

* AlertIsPresent()
* ElementSelectionStateToBe()
* VisibilityOfAllElementsLocatedBy()
* ElementToBeSelected()
* FrameToBeAvaliableAndSwitchToIt()
* InvisibilityOfTheElementLocated()
* InvisibilityOfElementWithText()
* ElementToBeClickable()
* PresenceOfAllElementsLocatedBy()
* PresenceOfElementLocated()
* TextToBePresentInElement()
* TextToBePresentInElementLocated()
* TextToBePresentInElementValue()
* TitleIs()
* TitleContains()
* VisibilityOf()
* VisibilityOfAllElements()
* VisibilityOfElementLocated()

# Accessing properties file

1. Create file (xxx.properties)
2. Give hardcode data as key value pairs (e.g., LoginId = Kalaiselvamk)
3. Calling from base class
   * Properties prop=new Properties();
   * FileInputStream fin = new FileInputStream (“properties file path”);
   * Prop.load(fin);
   * prop.getProperty(“LoginId”);

Commenting in properties file is “**#”**

If we call commented data the output will be “**null**”

# Frame handling

R-Click on frame 🡪 if frame option is available the frame exists.

All the frames are having the tag name as “iframe”.

To find the number of frames in the webpage

List<WebElement> frameList=driver.findElements(By.tagName(“iframe”));

Here we will not be able to find/locate elements inside frames.

For that we have to switch into frame.

driver.switchTo().frame(1); //pass frame number as parameter.

or

driver.switchTo().frame(“frame Name”); //pass frame name as parameter.

or

driver.switchTo().frame(“xpath of the frame”);

After that we can be able to handle elements

driver.findElement(//\*[@id=’username’]).sendKeys(“username”);

# Handling dynamic objects

**Dynamic Elements**

**If partially changes**

driver.findElement(By.xpath(“//\*[contains(@id,’username’)]”)).sendKeys(“username”);

driver.findElement(By.xpath(“//\*[starts-with(@id,’user’)]”)).sendKeys(“username”);

**If changes completely**

Selenium provides different API to use function keys. For example tab key, enter keys, F5 etc.

driver.findElement(By.id(“password”)).sendKeys(Keys.ENTER));

or

driver.findElement(By.id(“password”)).sendKeys(Keys.TAB));

# HTML Report/ Extend Reports

One default set of codes are there for listeners → copied in utilities package → also copy its corresponding XML code in TestNG XML hence it will listen all activities (immediately after suite tag ending)

ExtendReportListener

*“A set of code is available to generate Extend report, that implements* ***IReporter*** *interface, in that interface a method* ***generateReport()*** *will be invoked after each running suite have run.”*

# Retry/Retest

Two Levels

1. @Test Level
2. Run time Level

* IRetryAnalyser – Listener
* Counter=3;

*Create First class, that extends* ***IRetryAnalyser****:*

Public class **RetryAnalyser** implements **IRetryAnalyser**

{

int counter=0;

int limit=3;

public boolean **retry**(**ITestResult** result)

{

If(counter<limit)

{

return true;

}

return false;

}

@Test Level

@Test (retryAnalyser = Package.RetryAnalyser.*class*)

It will do retest if fails

Run Time Level

Create another class that extends ***IAnnotationTransformer***

Public class MyTransformer implements ***IAnnotationTransformer***

{

public void **transformer** (**ITestAnnotation** annotation, **Class** testClass, **Constructor** testConstructor, **Method** testMethod)

{

**Annotation**.setRetryAnalyser(**RetryAnalyser**.class)

}

TestNG coding for retry

<listeners>

<listener class-name = “package.MyTransformer”/>

<listeners>

Two Ways

1. Create RetryAnalyser class → implements ***IRetryAnalyser*** (From TestNG) → override **retry( )** method

Note: We can directly call this inside @Test annotations

1. Create MyTransformer class → implements ***IAnnotationTransformer*** (From TestNG) → override **transformer( )** method

→ call RetryAnalyser class (e.g., **Annotation**.setRetryAnalyser(**RetryAnalyser**.class))

Add <Listeners> code inside TestNG – It will retry the failed test cases by default

# Screenshot for failed TestCases

Create Listener class → implements ***ITestListener*** (From TestNG) → override **onTestFailure()** method

TestNG coding for Screenshot

<**listeners**>

    <**listener** class-name="com.pack.listeners.TestListener"/>

</**listeners**>

# Rerun Failed TestCases

@RunWith(ExtendedCucumber.**class**)

@ExtendedCucumberOptions (jsonReport = "target/cucumber.json", retryCount = 3)

Just add the above @ExtendedCucumberOptions above the @CucumberOptions and change the @RunWith(Cucumber.class) as @RunWith(ExtendedCucumber.class)

# Screenshot in Cucumber

@After → Screenshot code with the condition of Scenario.isFailed

# ExtendReport in cucumber

One plugin for com.vimalselvam dependency. Connect in runner file in cucumber options, in format

# Schedule Execution in Jenkins

Build periodically → so many options with regular expressions, by using those codes we can schedule.

# AutoIT code

class AUTOIT implements Runnable{

public void run()

{

Runtime.getRuntime().exec("path where AUTOIT cod exe file exists");

}

driver.get(“application URL”);

AUTOIT autoit=new autoit ();

Thread t1 =new Thread(autoit);

t1.start();

# Automation Testing

Test cases to be automated can be selected using the following criterion to increase the automation ROI

* High Risk - Business Critical test cases
* Test cases that are repeatedly executed
* Test Cases that are very tedious or difficult to perform manually
* Test Cases which are time-consuming

The following category of test cases are not suitable for automation:

* Test Cases that are newly designed and not executed manually at least once
* Test Cases for which the requirements are frequently changing
* Test cases which are executed on an ad-hoc basis.