AUTOMATION

Software Testing

Manual Testing

* Human involvement is necessary for design & Execution tests

Automation Testing

* Human involvement is only necessary for design tests and for execution no need of human.

ADVANTAGES:-

* Eliminate Human Errors.
* Reduce time for test Execution
* Perform repetation task with less effort.
* Tool Manage Execution 24/7.
* Perform repeated task

DISADVANTAGE:-

* Costly tool to use.
* If UI updated, we should update test code.

* Design of Automation test cases take more time compare to Manual design.

Build-1

Login → User
Password
Submit → Design Automation

Design script already passed test cases

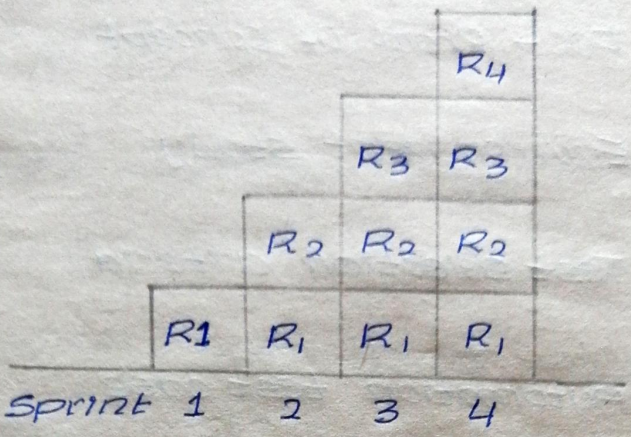
Build-2

Find Defect

Build-3

Find Defect

* Automation tools only manage regression testing.



Design and Execution of Automation cases sprint wise.

	Sprint 1	Sprint 2	Sprint 3	Sprint 4
Design	R_1	R_2	R_3	R_4
Execution	R_1	R_1, R_2	R_1, R_2, R_3	R_1, R_2, R_3, R_4

AUTOMATION LIFE CYCLE :-

POC → Proof Of concept

↓

Environment study

↓

Test selection criteria

↓

Automation test case

↓

Frame work Identification

[Developer/Sr. Automation test Engineer Ex :- 10+ years]

↓

· Basic script [According to framework]

↓

Design & Testing

Enhanced script [Loops, conditions, Reports, wait command]

↓

Debugging

↓

Execution [Jenkins]

↓ Script fail

* Re-design

↓ Test fail

* Raise a defect or Report

SELENIUM :-

* Selenium is a web interface automation tool.
* Selenium is designed to automate functional behaviour of software
* Selenium is a open source project owned by "GOOGLE".

SELENIUM COMPONENTS :-

IDE
webdriver
Grid

Selenium have set of components to automate browser Interface for each component it has different approach to automate browser interface

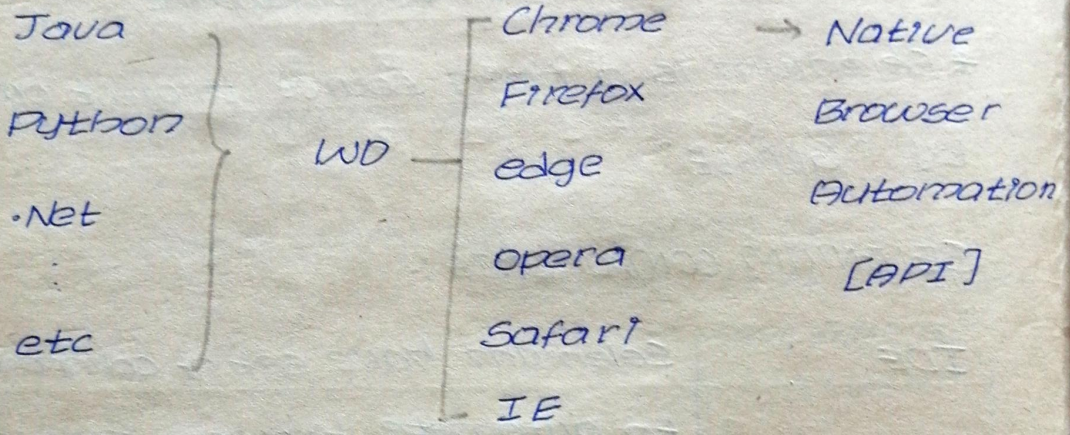
Selenium version 1 - Released in 2004.

IDE :- (Integrated Development Environment)

* IDE is extension for chrome and firefox.
* IDE support record and playback on webinterface.

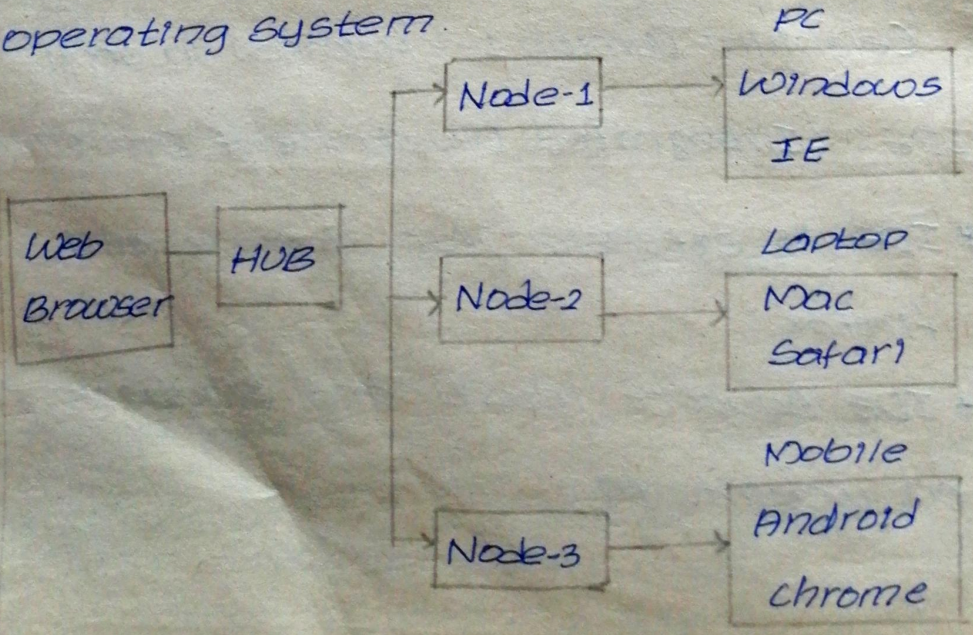
WEBDRIVER :-

Webdriver is not a tool it is a program API which supports webinterface automation on multiple browser.



GRID :-

Grid is a server which support parallel automation on multiple Browser and operating system.



SELENIUM ADVANTAGES :

* Support cross browser Automation
* Support multiple Languages to design test cases
* Support multiple operating system to install.
* Support mobile Interface Automation

Example :

Applications
- Native App
- Chrome App
- Browser App
- Hybrid App

* Support test Development driven frame-work. [TDD]

Example :

Testcase (Reporting & Runner)

* Support Behaviour development driven framework :

Example :

Cucumber → It helps to ~~design~~ develop test case in plane language

* Support Dynamic webpage handling
* Support API Automation.
* Support window handling feature Alerts, frames, Browser windows.
* Being a open source it has a good advantage that it will easily adopt any tools with in same platform.

SELENIUM IDE :-

* IDE is a plugin for chrome and firefox browsers.
* IDE is used to record and playback user interfaces.
* IDE assist in exploratory testing.

ADVANTAGES :-

* Default object identification support
* Recorded script is converted into different Languages.
* It can support automate small projects instead of using webdriver.
* Recorded script can run in different browser using command line feature

INSTALL Selenium IDE :

1. Open chrome
2. Visit selenium official website
   "http://selenium.dev.com//
3. Click on download button
4. Scroll down until you findout selenium IDE.
5. Under selenium IDE click on "for chrome" link.
6. At chrome addon page click on "Add to chrome" button.
7. Click Add extension to complete installation

Where to find Selenium IDE ?

1. You can find selenium (SE) icon in browser tool bar.
2. Click SE Icon to trigger selenium IDE.

RECORD AND PLAYBACK WITH IDE :

1. Open browser
2. Click on Selenium IDE at browser tool bar.
3. Click on "Record a new test in new project".
4. Enter project Name and click on ok button.
5. Enter valid URL and click start recording (This action take to required page.
6. Perform user actions at webpage using mouse and keyboard.
7. After completed user actions stop recording button.
8. Set Execution Speed to slow.
9. Hit play current test button
10. Correct any failures.
11. Save project after every test is successfully executed.
12. Click save project button and save project with .Side extension

WEB DRIVER :

Web Driver is a native browser automation API.

Note :- Because webdriver is a API, we need to configure environment before start automation tests.

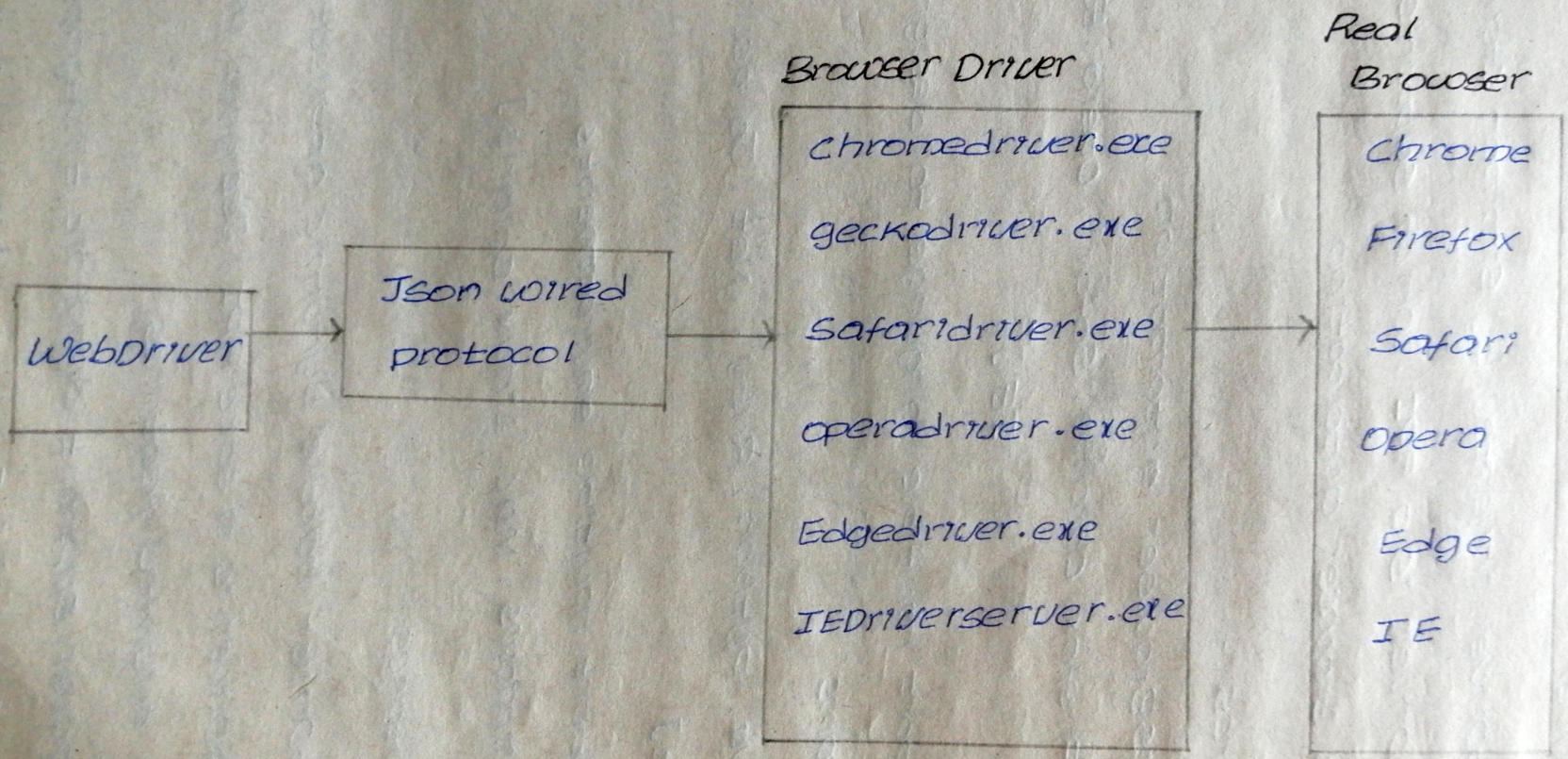
Downloads required for webdriver :-

1. JDK 1.8 (Java Development Kit)
2. Eclipse - IDE (Java development version)
3. Webdriver Libraries (Java language

* In eclipse place we can also use other IDE's

[intellij, Notepad's, Editplus, Netbeans. etc)

WEBDRIVER ARCHITECTURE :



WebDriver :-

* WebDriver invented by Simmon Stewart
* WebDriver is a API, which supports cross browser automation

Features :-

* CrossBrowser automation
  [Firefox, chrome, Safari, Opera, Edge, IE]
* Support multiple languages to design programs
  [Java, python, Ruby, .Net, Javascript etc]
* Support multiple environments
  [windows, Mac, Linux, Solaris]

Advantages :-

* WebDriver handle dynamic applications
* WebDriver is a core API, Designed with object oriented features.
* WebDriver support dynamic timegaps with Implict & Explicit Advanced waits

* WebDriver had partial support on Alert handling.
* Support mobile interface automation [Appium].
* Support BDD framework [Behaviour Development Driven] [cucumber]

Driver path KeyNames :-

Environment variable for chrome Brower :-

Keyname : webdriver.chrome.driver
Keyvalue : filepath\\chromedriver.exe

Setting runtime environment variable for chromedriver.exe

System.setProperty("keyname", "Keyvalue");

Environment variable for firefox Browser :-

Keyname : webdriver.gecko.driver
Keyvalue : filepath\\geckodriver.exe

Environment variable for Iebrowser :-

Keyname : webdriver.ie.driver
Keyvalue : filepath\\IEDriverserver.exe

Environment Variable for edge Browser :

Keyname : webdriver.edge.driver

Keyvalue : filepath \\edgedrive.exe

OBJECT IDENTIFICATION IN WEBDRIVER :

* In Selenium all components recognize objects using backend html source.

* To find Unique object properties we need to watch page html source using below options

    * Truepath [It generate relative xpath]

    * Default inspect option [All browsers]

Default inspect option at browser window:

    * Open Application

    * Press F12 (It opens html window)

    * From HTML window pick element at webpage

Navigation : Right click on element you wish to watch element html source

select → Inspection

→ It opens Html window along with selection of objects.

Dummy HTML object :

Attribute/property ↑ Attribute ↑ Attribute ↑

Account creation

↓

Text at front page

<> ⇒ Opentag → It contains tagname & Attributes / properties of object

</> ⇒ Closetag → It contains only tagname

Locators in Webdriver :

By is a locator class at webdriver. We have list of identification methods under By class.

1. id
2. name
3. classname
4. tagname
5. cssselector
6. xpath
7. linktext
8. partiallinktext

Note:- Findout any default webdriver locator identifiers matching in highlighed area and choose unique property before start script writing

id and name locators :-

id and name are default locators in html source. Developer set id and name for every object to recognize easier. Setting unique identification to objects is developer responsibility.

HtmlNode :-

```
<input id="x" class="ABC" type="email"
       name="User">
```

Syntax for name :-

```
driver.findElement(By.name("User"));
```

Syntax for id :-

```
driver.findElement(By.id("x");
```

Note:- To identify location otherthan ID & name property we follow xpath and cssselector. These selectors can identify objects by customize html sources.

HTML objects and Default / Unique Locators

object	Tagname	Default / Unique property
* textbox	<input>	type = "text" type = "email" or type = "password
* Link	<a>	href , title
* Dropdown	<select>	Under select List of <option> tag
* button	<button> <input> 	type = "Submit" or role = "button"
* checkbox	<input>	type = "checkbox"
* Radiobutton	<input>	type = "radio"
* image	<img>	src

* Unorderlist		contains lists of <Li> tags
* Orderlist		contains list of tags
* List of items	<Li>	Used under and
* header	<h> or <h1>	Header text
* division		Use for all objects to create order
* text	 <b>	NA
* icon	<i>	NA
* Label	<label>	Use to provide name for controls

How to Identify WebEdit box in Html source and prepare script for editbox :-

* For editbox developer set tagname as <input>.
* And define default property type as "text/password/email".

HTML source for editboxes :-

```
<input id="user1" type="email"
    class="inputbox" name="username">
```

Command to operate editbox :-

SendKeys :- SendKeys command useful to type characters at selected editbox.

Note :- SendKeys also useful to follow keyboard shortcut on selected location.

Clear :- Clear command is only for editboxes. It clear existing text at editbox, Because webdriver append characters in case text already available.

Syntax:

* Typing character sequence at editbox.

driver.findElement(By.id("User")).clear();

driver.findElement(By.id("User")).sendKeys

("Userinput"); (~~Keys.TAB~~).

* Keyboard inputs

driver.findElement(By.id("User")).sendKeys

(Keys.TAB);

* Character sequence and keyboard input

driver.findElement(By.id("User")).sendKeys

(Keys.TAB, "Userinput");

driver.findElement(By.id("User")).sendKeys

(Keys.TAB + "Userinput");

* Sequence of keyboard shortcuts to use:

driver.findElement(By.id("User")).sendKeys

(Keys.TAB, Keys.TAB);

Note:-

1. sendKeys command usefull to pass Ccharacter sequence or keyboard inputs7

2. clear command only we use for webedittbox because webdriver append text instead of overriding.

Identify Link at HTML Source:-

* In webpage links play keyrole and it load new page at browser window
* Link contains <a> anchor tag at backend html source.
* Usually developer set unique identification to link [href and tittle] properties.
* Only links to identify webdriver have seperate locator builders.

Linktext
partialLinktext

Linktext :- Locator identify only with original linkname.

partiallinktext :- Locator identify with original or partiallinkname.

Note :- Most of the time developer define link with href and title properties. Inorder to identify link with new href and title property we must use locator builders [xpath and css selector].

HTML Source :-

 Create Job

* Designing script to identify Link at webpage using Linkname.

driver.findElement(By.Linktext("Createjob")).
Click();

HTML Source :-

 Forgot Account

* Designing script to identify Link at webpage using Link name.

driver.findElement(By.Linktext("Forgot Account")).click();

HTML Source :-

 Selenium Tester
 In, Hyderabad

Note :- In above html source link name available in two different nodes. So that we can't use original linkname to identify link.

Identifying with partial Linkname :-

```
driver.findElement(By.partialLinkText
        ("Selenium Tester")).Click();
```

Why Clear() is Mandatory ?

* If webdriver identified other than textbox, clear command throws Exception [invalid state Exception].

* To avoid append text if already text exist.

→ without clear command :-

* When object duplicated webdriver give priority to help top object at Html tree

* If webdriver identified other than textbox, sendkeys pass keys on corong objects and continue to next line without throwing Exception

EXCEPTION :

* Illegal State exception

→ This exception display when driver path not setup (or) Invalid path setup to chromedriver.exe

→ Also get exception when keyname webdriver.chrome.driver with uppercase

* Session Not created

→ This exception present when browser driver mismatch with browser version.

* Invalid Argument Exception

→ This exception present when user missed http:// (or) https:// protocol

```
get("Http://Domainname.com");
get("http://instagram.com");      } Valid
get("https://facebook.com");

get("www.google.com");   Invalid
```

* WebDriver Exception :

Dri

How to identify dropdown at Htmlsource and select options from Dropdown:

Html source for dropdown:

```
<select id="country">
        <option value="Ind">India</option>
        <option value="Usa">America</option>
        <option value="UK">Kingdom</option>
</select>
```

Note:- At Htmlsource object contains select tag only we consider as dropdown or combobox, we use below select commands only for dropdowns.

1. SelectByVisibletext
2. SelectByValue
3. SelectByIndex

Syntax: to select dropdown option

```
//Instance selection method
new select(webElement).SelectByVisibleText(
                        "optionname");
```

Select class package:

Org.openqa.selenium.support.ui

Note: At webpages most of listboxes look like dropdowns, Those listboxes we can select by using click method.

* Selecting process for list boxes

Step1:- First click to open list of items

Step2: Second click to select one of item from List.

SelectByVisibleText:-

In dropdown fields All options have visible names, use option name to select any dropdown at webpage.

Syntax:

new select(webelement).selectbyvisibletext("optionName");

Html source:

<select id="month"> Visible text

index 0 <option value="1">Jan</option>
1 <option value="2">Feb</option>
2 <option value="3">Mar</option>

</select> Value

Example: Select feb option from above html source using Visible text

new select(driver.findElement(By.id("month"))
.selectByVisibletext("Feb");

Selectbyvalue :-

Get dropdown option value from html source, For every dropdown option developer set one unique value.

Syntax :-

```
new Select(webelement).Selectbyvalue(
                "Value in String");
```

Example :- Select option ~~United~~ Mar ~~Kingdom~~ from above source using selectbyvalue

```
new Select(driver.findElement(By.id("~~country~~month")))
                .selectbyvalue("3");
```

Select by Index :-

* In dropdown options index number always start with zero.
* Index of option-tag under select tag

Syntax :-

```
new Select(webelement).selectbyindex
                (index in numeric);
```

Example :- Select Feb option using index

```
new Select(driver.findElement(By.id("month")))
                .selectByIndex(1);
```

ismultiple :-

* Method useful to find dropdown selection state allow multiple or single option selection.
* ismultiple is a boolean return type it return <<< true >>> on multiple selection. and <<< false >>> on single option selection.

Syntax :-

```
boolean flag=new select(webelement).
                              ismultiple();
System.out.println(flag);
```

Deselection commands :-

These commands only works on multiple selection type dropdowns.

List of deselection commands :-

- deselectAll
- deselectByIndex
- deselectByvalue
- deselectBy Visibletext

DeselectByVisibletext :-

Syntax :-

```
new select(webelement).
        deselectbyVisibletext("optionName");
```

Deselect by index

Syntax :-

new select(webElement).deselectbyindex
(indexnumber);

Deselect by value :-

Syntax :-

new select(webElement).deselectbyvalue
("optionvalue");

Deselect All :-

Syntax :-

new select(webelement).deselectall();

Dropdown verification commands :-

getoptions

getAllselectedoptions

getFirstselected option

getoptions :- Method collect list of option tag elements under dropdown, and allowed user to verify expected options or optioncount under selected dropdown.

Example: How to get Dropdown options count

```
Select Dropdown = new Select (driver.findElement
                    (By.id("DropdownId")));
int obj_count = Dropdown.getOptions().size();
System.out.println("Object count available at
                    dropdown is" + obj_count);
```

getAllselectedoptions:

Note: This method only works on multiple selection dropdown

* Method collection list of options selected under multiple selection dropdown and its help verify selection count.

Example:- How to verify maximum selection count at dropdown.

```
Select Dropdown = new select (driver.findElement
                    (By.id("DropdownId")));
int selection_count = Dropdown.getAllselected
                    options().size();
System.out.println("Selected option count is"
                    + selection_count);
```

Note: Before use this command make sure dropdown is multiple option selection type.

Working with class Identifier :-

Note: In general class is to identify group of objects. But some times developer use class as unique property. so in that case we can use class as unique property to identify single objects..

Rules to follow while using class property:

* Check duplication of class before use.
* Don't use class property value along with spaces.
* Because developer use spaces with in class to define multiple class properties

Example:

By.className("navigation signin_link") → wrong

[Don't use classname with spaces particularly when using classname as property]

By.className("navigation");

[OR]

By.className("signin_links");

* Make sure duplication of class individually before use it.

Note : While defining class property along with xpath and cssselector locators we can include spaces.

By.xpath("//div[@class='navigation signin_link']");

By.cssSelector("div[class='navigation signin_link']");

CssSelector :-

cssselector is a global locator it recognize objects at UserInterface using backend html source.

Advantages of cssSelector :-

* WebDriver have inbuild identifiers [ID, Name, className, Tagname], with these identifiers we can't identify all objects at webpage. In this case cssSelector helps to identifies object

at webpage using other Locators as well [href, title, aria-label, type --etc]

* CssSelector identify locations using relative path

* CssSelector effectively identify objects at IE browser.

1. Default css
2. CSS-Attribute
3. Relative css [Axis specifier]
4. partial css
5. CSS-position

Default-CSS:-

Default css target object with only ID and class property [Optionally tagname]

htmlsource:-

```
<input id="email" class="Login"
                type="text"></input>
```

Using Default ID and classname:-

```
driver.findElement(By.id("email"));
driver.findElement(By.className("Login"));
```

Using CssSelector:-

```
driver.findElement(By.cssSelector("#email"));
driver.findElement(By.cssSelector(".Login"));
```

Note:

During Default cssSelector

→ Refer IDproperty

. → Refer Class property

Identify location with ID and class property combination:-

syntax:→ tagname#ID.classname

driver.findElement(By.cssSelector(

"input#email.Login"));

Example:

html source:

```
<div class="C"></div>
<div id="x" class="B"</div>
<div id="x" class="c"</div> → target Location
```

By.id("x") → 2nd

By.class("c") → 1st

By.cssSelector("div#x.c") → 3rd

Default selector combinations :-

#id

.classname

#id.classname

tagname#id.classname

tagname#id

tagname.classname

Default selector when class property having spaces :-

Html source :-

```
<button class="User reg account">
                                </button>
```

Identify with class property :-

```
By.className("User")
By.className("reg")
By.className("account")
```

Identify with cssSelector :-

```
By.cssselector(".user.reg.account")
By.cssselector(".User")
By.cssselector(".reg")
By.cssselector(".account")
By.cssSelector(".User.reg")
By.cssselector(".User.account")
By.cssselector(".reg.account")
```

CSS_Attribute:-

Instead of target elements with id and name property. we can target location with other attributes using Attribute syntax.

Note:- Make sure duplication for other properties before using it.

CSSAttribute Syntax:-

```
By.cssSelector("tagname[Attributename
                    ='Attributevalue']")

By.cssSelector("*[Attributename
                    ='Attributevalue']")
```

* While using different locators other than id & name property makesure those properties available unique.

Example:-

Html source:-

```
<div id="sbtn" aria-label="Login"
                    type="submit"></div>
```

By.cssSelector("#sbtn")

By.cssselector("div[id='sbtn']")

By.cssselector("div[aria-label='Login']")

By.cssSelector("div[type='submit']")

* We can also use star (*) instead of tagname

By.cssSelector("*[id='sbtn']")

Other than id unique attributes followed by developers :-

* Name
* aria-label
* placeholder
* dataid
* data-cy
* datatrackingid
* datareactid
* href
* title
* aria-describedby

CSS_Multiple Attributes :

Identify element with multiple attributes using cssselector.

syntax :- tagname [Pname = 'Pvalue']
[Pname = 'Pvalue']

CSSselector with classproperty :-

<Input class = "user login account">

By.cssSelector ("Input [class = 'user login account']

* During css attribute syntax we can define class property along with spaces

Dynamic CSS :- [cssSelector identify location with partial attributes]

To target Webelement location with partial attributes, cssSelector follow below regular expressions.

* → Any Matching characters

$ → End characters

^ → Start characters

Example 1 :- target element with matching characters

Manual browser → <input id="687dg-email-9526">
</input>

Automation browser →

<input id="734Cf-email-95472">
</input>

By.cssSelector("input[id*='email']")

Example 2 : target element with end char

MB → <input id="6943741-email"></input>

AB → <input id="7342161-email"></input>

By.cssSelector("input[id$='email']")

Example 3 : target element with start character

MB → <input id="email-7341264"></input>

AB → <input id="email-6142164"></inpt>

By.cssSelector("input[id^='email']")

Relative CSS :-

* Identify Location with parent node.

Syntax :-

```
By.cssSelector("Parenttagname[
        property='Pvalue']>Childtagname
```

Html source :-

```
<div id="reg-container" class="form">
    <input id="12ed345" type="text"></input>
                                  ↓
                          No unique property
</div>
```

```
By.cssSelector("div[id='reg-container']>input")
```

* Recognize object using parent Node when Element is duplicated.

Html source :-

```
<html>
<body>
    <div id="reg-container" class="form">
        <input id="email" type="text">
                                  </input>
    </div>
    <div id="reg-login" class="form">
        <input id="email" type="text">
                                  </input>
                                    ↓
                                  target
</body>
</html>
```

By.cssSelector("div[id='reg-login'] > input");

* Recognize object when parent element having Group child elements

Html source:-

```
<html>
<body>
  <div id="reg-container" class="form">
    <input id="email" type="text"></input>
  </div>
  <div id="reg-login" class="form">
    <input id="sample" class="abc"></input>
    <input id="email" type="text"></input>
                                  ↓
                                target
  </div>
</body>
</html>
```

By.CSS

Syntax:-

Parenttagname[property='pvalue'] >
childtagname[property='pvalue']

By.cssSelector("div[id='reg-login'] >
input[id='email']");

* Recognize object when parent having duplicate child elements

Html source :

```
<html>
<body>
  <div id="reg-login" class="form">
    <input id="sample" class="abc"></input>
    <input id="sample" class="abc"></input>
    <input id="sample" class="abc"></input>
    <input id="sample" class="abc"></input>
  </div>
</body>
</html>
```

CSSposition :

tagname : nth-child(int pos)

tagname : nth-of-type(int pos)

tagname : nth-last-child (int pos)

Relative Syntax with position :-

```
By.cssSelector("div[id='reg-login'] >
                input:nth-child(2)")

By.cssSelector("div[id='reg-login'] >
                input:nth-last-child(3)")
```

* Recognize element with position.

Html source :

```
<select id="classType" class="auto-complete">
        <option> Economy </option>
        <option> Business </option> → target
        <option> Peconomy </ option>
</select>
```

```
Select[id='classType'] > option:
                          nth-child(2)
```

* Identify object with sibling node

Html source :-

```
<div class = "abc"
    <input id = "firstname" class = "text">
    <label class = "abc" ></label>
    <div class = "abc"> </div> --> target
```

```
input[id='firstname'] + label + div

input[id='firstname'] ~ label ~ div
```

Note : In cssSelector + and ~ identify sibling nodes

Xpath :-

Xpath is a native location identification at webpages. Xpath Identify locations at webpages using backend html source.

1. Abcolute xpath
2. Relative xpath

Xpath Attribute :-

Below syntax identify location at webpage using unique attributes.

//tagname[@Attributename='Attribute value']

//*[@Attributename='Attribute value']

// → Current Node

/ → RootNode

tagname → only selected tag element

* → All elements under current node

[] → Allow multiple attributes to define

@ → only Attributes → Refer attribute at html node

Html source:-

<input id="email" class="user_email_login"

placeholder="Enter name"></input>

//input[@id='email']

//input[@class='user_email_login']

//input[@placeholder='Enter name']

Xpath text:-

Below syntax identify location using element text.

//tagname[.='Element Text']

//tagname[text()='Element Text']

text() → Refer element text

. → Refer element text

Html source:-

Create Account

//span[text()='Create Account']

//span[.='Create Account']

Dynamic Xpath :

Identify location using partial attributes or text.

* Recognize object using matching characters

//tagname[contains(@prop, 'match characters')]

//tagname[contains(text(), 'partial visibletext')]

* Recognize object using partial attribute or text using start characters

//tagname[starts-with(@prop, 'pvalue')]

//tagname[starts-with(text(), 'text start chars')]

* Recognize object using partial attribute or text using end characters

//tagname[ends-with(@prop, 'pvalue')]

//tagname[ends-with(text(), 'text ends chars')]

Example : Target Element with partial attribute properties

Manual Browser session :

id="email-987fsd15jks

Automation Browser session :

id="email-732dsgfoskl

//*[contains(@id, 'email')]

To identify xpath using AND function :-

//tagname[@prop='pvalue and @prop='pvalue']

//tagname[@prop='pvalue and text()='innertext']

To identify xpath using OR function :-

//tagname[@prop='pvalue or @prop='pvalue']

//tagname[@prop='pvalue or text()='innertext']

To identify object with multiple Attributes [using <OR> <And> comparision] :-

//*[@prop='pvalue' and (@prop='pvalue' or @prop='pvalue')]

//*[@prop='pvalue' or (@prop='pvalue' and @prop='pvalue')]

Example : Target Element with parent Node.

Html source :

```
<div id="login-container " class="demo">
<input id="qju7y9876" type="text">
```

↓ target

```
</div>
```

Syntax :

Parentnode / childnode

//div[@id='login-container']/input

Xpath Duplicate index :

(Any xpath) [int position]

For the duplicate properties we can provide index number..

Html source :

```
<div class="sbtn">
    <input type="text">
</div>
<div class="sbtn">
    <input type="text">
</div>
```

(//input[@type='text'])[2]

Recognize object using following sibling :-

Syntax :-

//SiblingElement/following-sibling::tagname

Html source :-

```
<div class="origin">
    <input id="firstname" name="new">
    <div class="origin"> → target
</div>
```

//input[@id='firstname']/following-sibling::div

Recognize object using preceding sibling :-

Syntax :-

//SiblingElement/preceding-sibling::div

Html source :-

```
<div class="origin">
    <div class="origin">
    <div class="roundtrip">
    <div class="origin"> → target
    <div class="oneway">
    <input id="firstname" name="new">
</div>
```

//input[@id='firstname']/preceding-sibling::div[2]

Recognize parent element with child element :-

Syntax :-

//childelement/parent::tagname

Html source :-

 → target

<input id="firstname" name="new">

//input[@id='firstname']/parent::div

Recognize descendant element using ancestor elements :-

Syntax :-

//uniqueparent/descendant::childtagname

Html source :-

 → unique

<input id="firstname"> → target

//div[@class='origin']/descendant::input

Recognize ancestor element using descendent elements:

Syntax :

//uniquechild/ancestor::parent tagname

Html source :

```
<ul class="origin">        → target
    <div class="origin">
        <input id="firstname"> → unique
    </div>
</ul>
```

//input[@id='firstname']/ancestor::ul[1]

User Interface Verification commands :-

In webdriver all verification commands starts with << get >>. keyword. These commands work like checkpoints, which helps to verify expected behaviour presented at webpage.

These commands categorized into two types.

1. Page verification commands
2. Element Verification commands

1. Page Verification commands :-

* getTitle
* getCurrentUrl
* getPageSource
* getWindowHandle
* getWindowHandles

Syntax :-

```
driver.getTitle();
```

2. Element Verification commands :

* getText
* getAttribute
* getLocation
* getSize
* getTagname
* getRect

Syntax :

driver.findElement(By.id("id")).getText();

3. Dropdown Verification commands :

* getOptions
* getFirstSelectedOption
* getAllSelectedOptions

Syntax :

new select(webElement).getOptions();

getTitle :-

Method capture current window title at runtime. In webapplication every page contain independant title. By capture runtime title we can verify expected page displayed at browser window

Syntax :

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

getCurrenturl :-

Method capture current window url at runtime. In webapplication every page contains unique URL. By capture runtime url we can verify expected page displayed at browser window.

Syntax :

```
String Runtime_url =driver.getCurrentUrl();
System.out.println(Runtime_url);
```

getPageSource:

Method capture entire pageSource into string format during runtime. [backend HTML source].

With any source we can verify

- Element ID's & HTML Nodes
- Title of a page
- Url of a page
- Page text

Syntax:

```
String pageSource = driver.getPageSource();
System.out.println(pageSource);
```

getText:

Method capture visible text using location from the backend it capture innertext from the selected node.

⇒ It capture selected node and child node text as well.

Syntax:

```
WebElement Element = driver.
                findElement(By.id("x"));
String Element.text = Element.getText();
System.out.println(Element.text);
```

Note1: if selected element not contains any text, method return empty status

Note2: Method capture selected element and it's child innertext.

Note3: Method capture only visible text on webpage, In case element is hidden getText return empty status

What can we verify with gettext?

→ we can verify status message visible at webpage.

→ we can verify error messages visible at webpage.

→ we can verify expected text visible at webpage.

getAttribute ("Propertyname"):-

Method capture runtime attribute value using attributename.

Syntax:-

```
WebElement Element = driver.
        findElement (By.id("x"));
String value = Element.getAttribute
        ("attributename");
System.out.println (value);
```

Note: Developer use attributes, to change runtime behaviour of element. by capture element attributes at runtime we can prove element behaviour available w.r.t client requirement.

```
<input id="password" style="display:none
        type="text">
WebElement Email = driver.
        findElement (By.id("email"));
String Rvalue = Element.getAttribute("style");
System.out.println(Rvalue); →
        display:none
```

```
<input id="password" disabled type="text">
WebElement password = driver.
                findElement(By.id("password"));
String RDvalue = password.getAttribute
                ("disabled");
System.out.println(RDvalue)   → true
```

```
<input id="email" readonly type="email">
WebElement Email = driver.
                findElement(By.id("email"));
String REvalue = Email.getAttribute("class");
System.out.println(REvalue);   → true
```

How to capture input from editbox :-

```
WebElement Editbox = driver.
                findElement(By.id("EditboxID"));
String InputValue = Editbox.getAttribute
                ("value");
System.out.println("Inserted input values
at editbox is → " + InputValue);
```

getcssvalue :

Method capture selected element styles at webpages.

- font
- font-size
- color
- background-colour
- text-alignment

Syntax:

```
WebElement Element = driver.
                findElement(By.id("x"));
String text_font-size = Element.
                getCssValue("font-size");
System.out.println(text_font_size);
```

getLocation :

Method capture selected element location at webpage and return x and y coordinates of object

Note: getLocation return type is point. point return object x and y coordinates

Syntax:

```
WebElement Element =driver.
                    findElement (By.id("x"));
Point obj-point =Element.getLocation();
//Using object point get x and y coordinates
int obj-x = Obj-point.getX();
s.o.p("object x coordinates "+obj-x);
int obj-y = obj-point.getY();
s.o.p("object y coordinates "+obj-y);
```

getsize:

Method capture selected element dimension at webpage and dimension return height and width of object.

```
WebElement Element = driver.
                    findElement(By.id("x"));
Dimension obj-dimension = Element.getsize();
//Using object dimension get object height and width
int obj-height = Obj-dimension.getheight();
int obj-width = obj-dimension.getwidth();
```

Get Rectangle commands :

```
Webelement.getHeight();
Webelement.getWidth();
Webelement.getx();
Webelement.gety();
```

Program to get system default time :

```
Date d=new Date();
        //import java.util.Date
```

Create simple date format :

```
SimpleDateFormat sdf =
        new SimpleDateFormat("dd/MM")
```

Convert default system date using simple date formatter :

```
String sDate = sdf.format(d);
```

Date formatters :

YYYY ⇒ Year [2020]

YY ⇒ Year [20]

MM ⇒ Month [01-12]

MMM ⇒ Month [Jan-Dec]

EEE ⇒ Week [Sun-sat]

dd ⇒ date [01-31]

hh ⇒ hour [01-24]

mm ⇒ minute [01-60]

ss ⇒ seconds [01-60]

User Interface validation commands :-

All validation commands return boolean value true/false. These methods validate expected behaviour at webpage and return result in boolean format.

IsDisplayed

IsEnabled

Isselected

Drop down validation method

Ismultiple()

isDisplayed :-

Method verify element visibility or hidden state and return boolean value true/false during runtime.

* Method return true when selected element visible at webpage.

*Method return false when selected element hidden at webpage.

Syntax:

```
WebElement Element =
        driver.findElement(By.id("x"));
boolean flag = Element.IsDisplayed();
```

IsEnabled:

* Method Verify element enabled or disabled status.
* Boolean return type method, only return true/false.
* Method return true when selected element ready state is ON.
  [Element is enabled and accept all type of actions].
* Method return false when selected element ready state is OFF
  [Element is disabled and not accepting anytype of actions].

Syntax:

```
WebElement Element =
    driver.findElement(By.id("x"));
boolean flag = Element.IsEnabled();
```

Isselected:

Method return true/false on element selection state for radio button and checkboxes.

Syntax:

```
WebElement Element =
    driver.findElement(By.id("x"));
boolean flag = Element.IsSelected();
```

SWITCH COMMANDS:

Alert:

Developer most commonly use alerts at webpage to display info messages. these alerts disable page until it closed.

Types of alerts :-

1. Alert popup window
2. confirmation alert window
3. prompt window

How to find alert at webpage :

* Alert display info message in seperate popup window, and when alert presented user can't access other elements at webpage until alert closed

* Using exception also we can identify alert presented or not presented at webpage.

1. UnhandledAlert Exception
2. NoAlertpresented Exception

Alert exceptions in selenium :-

1. UnhandledAlert Exception :-

Exception present when alert was not handled at webpage.

2. NoAlertPresented Exception :

Exception present when expected alert not presented prompt at webpage

accept :

Method close alert at webpage

syntax :

driver.switchto().accept();

dismiss :-

Method dismiss/accept alert at webpage

syntax :

driver.switchto().dismiss();

getText :

Method capture text presented at alert

syntax :

String alert_msg =

driver.switchto().alert().getText();

S.o.p(alert_msg);

SendKeys :

Method send characters at editbox inside alert window.

```
driver.switchto().alert().sendkeys(" ");
```

Alert Handling Using Expected condition :

```
if(ExpectedConditions.alertIspresent().
        apply(driver) != null)
{
   S.O.P(" Alert presented ");
   // Do further actions if needed.
}
else
{
   S.O.P(" Alert is not presented ");
}
```

Alert Handling Using try catch :

```
try {
    S.O.P(" Alert presented ");
    // Do further actions needed
  } catch (NoAlertPresentedException e){
     S.O.P (" alert not presented");
  }
```

Frame :-

* A frame is a part of a webpage or browser window

* Frame has ability to load content independent like webpage.

* In selenium under frame source to identify, First we need to switch controls from webpage to frame.

How to identify frame at webpage ?

* When user right click on element, At context menu it display Thisframe/ View frame source options.

* We can find a frame by search under selector [xpath / cssSelector]

xpath :- //iframe [or] //frame

cssSelector :- iframe [or] frame

* We can also find frame using Truepath plugin.

→ For frame under elements truepath plugin generate seperate xpath.

Html source for frame element :-

```
<html>
<body>
    <div id="reg">
        <div id="UID">
            <input id="email">
        </div>
    </div>
    <frame id="newframe" name="f1"
                class="reg.account">
        <html>
            <body>
                <div id="login">
                    <div id="frame">
                        <input id="firstname">
                    </div>
                </div>
            </body>
        </html>
    </frame>
</body>
</html>
```

Element to target under frame

Frame Syntax :

* When frame have ID and Name Property

```
driver.switchTo().frame("ID/name");
```

* When frame have different properties other than ID/name.

```
driver.switchTo().
  frame(driver.findElement(By.ClassName("x")));
```

* Switching to frame Using index number

```
driver.switchTo().frame(5);
```

How to switch one frame to another frame :

1. Switch to FirstFrame

```
driver.switchTo().frame("ID");
```

2. Get control from firstframe to mainpage

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

3. Apply switch to second frame

```
driver.switchTo().frame("ID");
```

How to switch to parent frame :-

Under parent frame switch required between two sub frames. To get controls to parent frame we follow below syntax.

driver.switchTO().parentframe();

Window :-

How to switch to next window :-

```
driver.get("https://facebook.com");
String WIN1_ID=driver.getWindowHandle();
 // Gets current window dynamic id

WebElement Instagram.hyper_link
  =driver.findElement(By.linktext("Instagram"));
 Instagram_hyper_link.click();
```

* Below Method retrieve all window dynamic id's which was open through webdriver session.

```
Set<String> All_WIN_IDS =
        driver.getWindowHandles();
```

* Apply for each loop to read collection of ID's.

```
for (String SingleID : All_WIN_IDs)
{
  driver.switchTo().window(SingleID);
  }
```

* Above loop shift Access from facebook to instagram. Now we can perform actions in instagram window.

Switch to required window with page title:-

```
for(String SingleID : All_WIN_IDs)
{
  driver.switchTo().window(SingleID);
  String Runtime_title = driver.getTitle();
  if(Runtime_title.contains("Instagram"))
   {
     break;
   }
}
```

* Switch to window without using loop :

Switch to window using Iterator :

```
Set<String> All-window-IDS =
        driver.getWindowHandles();

Iterator<String> itr = All-window-IDs.
                        iterator();
```

* Target each token of Iterator value using "Next" keyword

```
String WindowID1 = itr.next();
String WindowID2 = itr.next();

driver.switchTo().window(WindowID2);
```

→ we can perform actions in window 2

```
driver.switchTo().window(WindowID1);
```

→ we can perform actions in window1

Active Element :-

Sometimes Selenium fail to identify ajax list due to different property architecture. In that case we can use activeElement method to target object which is focused at webpage

```
driver.switchTo().activeElement().sendKeys("
```

INTERACTIONS :

Mouse Interactions :

In webdriver all mouse interaction commands available at "Actions" class.

1. Click [It Analog left click at mouse]
2. Doubleclick
3. contextclick [It Analog right click at mouse]
4. MovetoElement
5. ClickAndHold
6. Release
7. DragandDrop
8. DragandDropBy

Action Methods :

1. Pause
2. Build
3. Perform

Note : Inorder to use mouse interactions, we should create object for << Actions >> class.

Actions actions = new Actions(driver);

Click :

Method analog left click action using mouse.

→ Mouse left click action at webpage

Syntax :

```
actions.click(element).perform();
```

MovetoElement :

Method analog hover action at element

Syntax :

```
actions.movetoElement(element).Perform();
```

Double click :

perform double click on selected target/webpage.

Syntax :

```
actions.doubleclick(element).perform();
```

ContextClick :

Method analog right click action at mouse

Syntax :

```
actions.contextclick(element).perform();
```

ClickAndHold

Method hold mouse left click button down. And allow use drag mouse to any expected target at webpage.

[Text selections at webpage]

[Group of option selections]

[Drag object from source location to any destination at webpage]

Syntax:

```
Actions.clickAndHold(WebElement src).
moveToElement(WebElement dst).release().
perform();
```

Note: We should use clickAndHold & moveToElement & Release as combination inorder to perform any single action.

DragAndDrop:

Method allow user to drag source element to target location.

Syntax:

```
WebElement src=driver.findElement(By.id("x"));
WebElement Dst=driver.findElement(By.id("y"));
Actions.DragAndDrop(Src, Dst).perform();
```

DragAndDropBy :

Method allows to drag any single element to particular x and y coordinates at webpage

Syntax :

```
WebElement src =driver.findElement
                    (By.id("x"));
actions.DragAndDropBy(src, 50, 70).Perform;
```

Action methods :

Perform :

Without perform method all mouse and keyboard interaction are incomplete.

Build :

Any group of mouse and keyboard actions to build as single option

Syntax :

```
actions.click(WebElement).click(WebElement)
        build().perform();
```

Pause :

While using any mouse actions we can send interval timings using milliseconds.

Keyboard Interactions:

These interaction commands we can find under << actions >> class.

1. SendKeys
2. KeyDown
3. KeyUP

SendKeys:

Accept character sequences and keyboard shortcuts.

Syntax:

```
actions.SendKeys(Element.Keys.TAB).
                        perform();

actions.SendKeys(Keys.ESCAPE).perform();
```

Keydown:

Usefull for doonkeys at keyboard shortcuts.

Syntax:

Use keyboard shortcut Ctrl+Alt+T

```
actions.keydown(Keys.CONTROL).
keydown(Keys.ALT).sendKey(Keys.chord("T").
                        perform();
```

ROBOT :

* Robot is a Java automation framework from [AWT]

* It support automation on native inputs [Keyboard and mouse]

* Robot can manage set of actions with mouse and keyboard at any interface.

Where to Implement robot framework in Selenium :

To handle window interface automation

[During file uploading and downloading time we must interact with window system files..]

Other window automation tools :

SIKULI → Image Recognition Automation tool.

AUTOIT → Window interface automation tool.

Robot Mouse commands :-

```
Robot robot = new Robot();
robot.setAutoDelay(1000);
```

* Command to move curser to required location

```
robot.mouseMove(100,200);
```

* Mouse click commands

```
robot.mousePress(InputEvent.BUTTON1_MASK);
                                    //Left click
robot.mousePress(InputEvent.BUTTON2_MASK);
                                    //middle click
robot.mousePress(InputEvent.BUTTON3_MASK);
                                    //Right click
```

* Mouse release

```
robot.mouseRelease(InputEvent.BUTTON1_MASK);
```

* Scroll

```
robot.mouseWheel(100);
```

Robot keyboard commands :-

robot.keypress (KeyEvent.VK_H);

* We have to perform KeyRelease actions for down keys.

robot.keypress (KeyEvent.VK_CONTROL);

robot.keypress(KeyEvent.VK_V);

robot.keyRelease(KeyEvent.VK_CONTROL);

Copy Paste String with robot Interface:

String path="C:\\Users\\SUNIL\\Desktop\\
Resume.docx";

*copy string

StringSelection Spath = new StringSelection(path

* Enable clipboard

Clipboard clipboard =

Toolkit.getdefaultToolkit().getSystemClipboard()

* Copy content to clipboard

clipboard.setContents(Spath, Spath);

CAPTURE SCREEN :-

* Create screenshot and convert into file format

```
File src = ((TakeScreenshot)driver).
        getScreenshotAs(OutputType.FILE);
```

* Create a new folder

```
FileHandler.createDir(new File("Screenshots"));
```

* Save File in a folder

```
FileHandler.copy(src, new File
        ("screenshots\\image.png"));
```

Disadvantage :-

* In selenium 3 version screen only capture visible interface.

How to overcome this issue :-

* In selenium 4 same command capture entire page.
* Using mouse interaction or keyboard interaction navigate to required interface inorder to capture screen.

Capture screen with time stamp :

Selenium override screen inorder to create new screen we had to add time stamp to file name.

* Get current system ~~dat~~ time

```
Date d = new Date();
Dateformat df = new SimpleDateFormat
                ("yyyy-MMM-dd hh-mm-ss");
String time = df.Format(d);
```

* Take screenshot

```
File src = ((TakeScreenshot)driver).
           getScreenshotAs(OutputType.FILE);
FileHandler.copy(src, new File
           ("Screenshots\\"+time+"naukri.png");
```

Capture screen when alert presented :

* Using Robot interface capture alert

Get screen dimension

```
Dimension Screen_Dimension =
  Toolkit.getDefaultToolkit().getScreenSize();
```

* Capture screen

BufferedImage image = robot.
createScreenCapture(new Rectangle(Screen.Dimension

* Save into local utilities

File file = new FILE ("file location \Valert.png");

ImageIO.write(image, "PNG", file);

Image extension :-

PNG :- portal network graphic

bmp :- bitmap image

JAVASCRIPT EXECUTOR :-

* Javascript executor is a interface class. It allows to make runtime changes at automation browser partially.

* Interface class inject original javascript into runtime browser to automate web interfaces.

When to use Javascript :-

We use javascript only when we get challenges only.

Because of cross domain policies browsers enforce your script execution may

fail unexpectedly and without adequate error messaging. mostly happened when trying to access another frame.

Note :-

Most times when troubleshooting failure it's best to view the browser's console after executing the webdriver request.

Handling action commands using Javascript ;

```
JavaScriptExecutor JS =
        ((JavaScriptExecutor) driver);
```

* Type text into editbox

```
JS.executeScript("document.getElementById
        ('email').value = 'darshan@gmail.com'
```

(OR)

```
WebElement email = driver.findElement
        (By.id("email"));
JS.executeScript("arguments[0].value = 'darshan
                @gmail.com'", email);
```

* Clicking buttons using JS

```
WebElement Login-btn = driver.findElement(
                By.name("login"));
JS.executeScript("arguments[0].click();", Login_btn);
```

* Selecting dropdown options

```
JS.executescript("document.getElementById
                  ('day').value = '6' );
```

* Selecting radio buttons

```
JS.executescript("arguments[0].click();",
                                  element );
```

Note: We can also use "checked = true" in place of click :

Highlighting object Using Javascript:

```
JS.executescript("arguments[0].style.
                  backgroundcolor = 'red' ",element);
```

* Runtime actions with Javascript

```
arguments[0].disabled = true/false
arguments[0].style.visibility = 'visible'/'hidden'
```

* Scroll to view using Javascript

```
arguments[0].scrollIntoView()
```

* Scroll methods

```
JS.executescript("window.scroll(0,300)");
```

WAIT COMMANDS :

Also known as synchronization commands.. Webdriver had default synchronization support on new pageload For other timeouts to manage we follow below wait commands.

Thread.sleep(ms);

ImplicitWait

ExplicitWait

FluentWait

Thread.sleep(ms) :-

is a Java timeout command it won't specify timegap w.r.t application

```
Public void method1() throws InterruptedException
{
  Thread.sleep(5000);
}
```

Note: Thread.sleep is a exceptional command to avoid runtime syntax error. Add throws InterruptedException to method.

Implicitwait :

* Implicitwait manage timegap w.r.t application.
* Implicitwait we define only once at browser launch time.

Implicit wait mainly categorized into 3 types

Implicitlywait

pageload Timeout

setscriptTimeout

* Implicit wait enable timeout on Exception only. After enabled timeout once expected behaviour found it continue execution without wait for timeout.
* Incase Expected behaviour not found until timeout completed. it throws exception

Syntax :

*Set timeout until object load at document object model [DOM]

driver.manage().timeouts().

Implicitlywait(50, Timeunit.Seconds);

* Set timeout until asynchronized source to load.

driver.manage().timeouts().

setScriptTimeout(50, TimeUnit.SECONDS);

* Set timeout until pageobjects load at browser window.

driver.manage().timeouts().

~~setScri~~ pageLoadTimeout(50, TimeUnit.

seconds);

Explicitwait :-

* Explicitwait manage timegap w.r.t application.

* Using explicitwait statement we can set timegap for object behaviour. Explicitwait waits until object behaviour load at front page.

Note: Explicitwait accomplish with "Expected conditions" class

ExpectedConditions :-

is a UserInterface Validation class. It contain set of validation commands to check UserInterface behaviour.

Note: Expected conditions also can implement directly without help of wait commands.

* titleis
* titlecontains
* Urltobe
* Urlcontains
* ElementselectionstateToBe
* ElementtobeSelected
* VisibilityofElementLocated
* InvisibilityofElementLocated
* PresenceofElementLocated
* ElementtobeClickable
* TextpresentedAtElementLocated
* TextpresentedAtElementValue
* AttributeContains

* AttributeTobe
* AlertPresented
* FrameTobeAvailableAndSwitchToIt
* NumberofWindowsTobe

Syntax:

```
WebdriverWait wait=
        new WebDriverwait(driver,100);
wait.until(ExpectedConditions.
    titleContains("Expected page title"));
```

ExpectedConditions:

titleis: An expectation to verify expected title presented at webpage

Syntax:

```
wait.until(EnpectedConditions.
        titleis("Expected page title"));
```

titleContains: An expectation to verify expected or partial title presented at webpage.

syntax:

```
wait.until(ExpectedConditions.
        titlecontains("title or partial title"));
```

CHROME OPTIONS :-

Headless-Browser :-

End of the day selenium script need to run using CI tool [continuation Integration tools] like jenkins. These CI tools run selenium in background, there this headless browser options is very usefull.

Syntax:

```
ChromeOptions options=
        new ChromeOptions();
options.setHeadless(true);
```

* Alternate method to set Headless argument

```
options.addArguments("--headless");

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

* Window maximize using chromeoptions:-

```
Options.addArguments("start-maximized");
```

Adding plugins in Automation Browser:

To add plugins we need to download crx files of that plugin.

https://www.crx4chrome.com

```
options.addExtensions(new File("filelocation"));
```

Chrome incognito mode:

```
options.addArguments("incognito");
```

Chrome option to disable popup blocker:

```
Options.setExperimentalOption
("excludeSwitches", Arrays.asList(
        "disable-popup-blocking"));
```

Chrome window set size:

```
options.addArguments("window-size=600,400");
```

BROWSER NAVIGATION COMMANDS :-

Note :- Selenium doesn't automate browser options, only ~~operate~~ automate page interfaces with in browser.

To manage basic browser option we have few navigation commands available

back() :→

Move back to previous page from current window

Syntax :-

```
driver.navigate().back();
```

forward() :-

Move a single page forward in the browser's window.

Syntax :-

```
driver.navigate().forward();
```

refresh():

Refresh the current page

Syntax:

driver.navigate().~~forward()~~;
refresh();

to(URL):

Load url to browser window using navigate command.

Syntax:

driver.navigate.to(URL);

Common Browser actions:

close(): Method close single browser window opened by webdriver

Syntax:

driver.close();

quit(): Method close all browser windows opened through webdriver

Syntax:

driver.quit();

Submit(): Method is useful to submit forms and submit action on location [It is similar to press enter key on webpage].

Syntax: driver.findElement(By.id("x")).Submit