**Window-handles in selenium:**

**Parent window** : The main window on which the user has currently landed or webpage on which the user is landed after hitting a link/URL. Such a webpage in Selenium is referred as the parent window also known as the main window.

**Child Window** : All the windows which will open inside your main window will be called as child windows.

**Window Handle**: A window handle stores the unique address of the browser windows. It is just like window navigator , whose return type is alphanumeric(**String**). The window handle in Selenium helps in handling multiple windows and child windows. Each browser/tab will have a unique window handle value with which we can uniquely identify it.

1. getWindowHandle() : This method will handle the **current window** that uniquely identifies it within this driver instance. Its return type is **String**.

Syntax :

String windowHandle = chrome.getWindowHandle();

System.out.println("Parent window Handle : "+windowHandle);

1. getWindowHandles() : To handle all **opened windows** by web driver, we can use “getWindowHandles()” method and then we can **switch** window from one window to another in a web application. Its return type is Set<String>

Syntax:

Set<String> windowsHandles = chrome.getWindowHandles();

for(String window:windowsHandles)

{

System.out.println(window);

}

**Alerts in selenium:**

**What is an alert?**

Alerts are small popup boxes/desktop-windows which display the messages/notifications and notify the user with some information seeking some permission on certain kinds of operations. Additionally, we can also use them for warning purposes. Sometimes, the user can enter a few details in the alert box as well.

**Types of alerts provided by Web Applications:**

**1.Simple alert**: These alerts are just informational alerts and have an OK button on them. Users can click on the OK button after reading the message displayed on the alert box. A simple alert box looks like below.

Graphical user interface, text, application, email

Description automatically generated

**2.Prompt Alert**: In Prompt alerts, some input requirement is there from the user in the form of text needs to enter in the alert box. A prompt alert box is displayed like below, where the user can enter his/her username and press the OK button or cancel the alert box without entering any details.

Graphical user interface, application, website, Teams

Description automatically generated

**3.Confirmation Alert**: These alerts get some confirmation from the user in the form of accepting or dismissing the message box. They are different from prompt alerts in a way that the user cannot enter anything as there is no text-box available. Users can only read the message and provide the inputs by pressing the OK/Cancel button.

Graphical user interface, application

Description automatically generated

accept() : This method is used to click on accept/ok/yes button on an alert.

Dismiss() : This method is used to click on cancel/no/reject button on an alert.

getText() : This method is used to return the text present on an alert.

sendKeys() : This method is used to send some commands to an alert.

**Syntax:**

WebDriver chrome = new ChromeDriver();

chrome.switchTo().alert().accept();

chrome.switchTo().alert().dismiss();

chrome.switchTo().alert().sendKeys("Hello Siddhi");

String alertText = chrome.switchTo().alert().getText();

System.out.println(alertText);

**IFrames in selenium:**

**What is iframe ?**

* + Iframe is webpage which is embedded in another web page

**(or)**

* + It is a HTML document embedded inside another HTML document
  + These frames are used in a webpage to add content from other source like advertisements in a webpage.
  + These iframe are defined with **<iframe>** tag.

**How to identity iframe on a web page?**

* We cannot detect the frames by just seeing the page or by inspecting on webpage.
* But when we try to inspect on web page, on context click options it will display

**View frame source**  or **Reload frame** in chrome browserand **This Frame** in Firefox bowser by these options we can confirm that webpage consists of frames.

Graphical user interface, text, application, email

Description automatically generated 

**frame(int )** : This method is used to switch the frame by its index.

**frame(String)** : This method is used to switch the frame by its name or id.

**frame(WebElement)** : This method is used to switch the frame by its webelement.

**parentFrame()** : This method is used to switch parent frame from child frame.

**defaultContent()** : This method is used to switch parent page from any frame.

**Single Frame:**

Graphical user interface, application, Word

Description automatically generated

**Html code:**

<html>

<head> </head>

<body class=" darktheme">

<iframe frameborder="20" id="frame1" name="frame1" allowfullscreen="true">

<button type="submit1" id="loginbutton” form="nameform1" value="Submit">Submit1</button>

</iframe>

</body>

</html>

**Selenium code :**

**With index:**

chrome.switchTo().frame(0);

chrome.findElement(By.id("loginbutton")).click();

chrome.switchTo().defaultContent();

or

**With id or name:**

chrome.switchTo().frame("frame1");

chrome.findElement(By.id("loginbutton")).click();

chrome.switchTo().defaultContent();

or

**With web-element:**

chrome.switchTo().frame(chrome.findElement(By.id("iframe1")).);

chrome.findElement(By.id("loginbutton")).click();

chrome.switchTo().defaultContent();

**Multiple frame:**

Diagram

Description automatically generated

**Html code:**

<html>

<head> </head>

<body class=" darktheme">

<iframe frameborder="0" id="frame1" name="frame1" allowfullscreen="true">

<button type="submit" form="nameform" value="Submit">Submit1</button>

</iframe>

<iframe frameborder="0" id="frame2" name="frame2" allowfullscreen="true">

<button type="submit" form="nameform" value="Submit">Submit2</button>

</iframe>

<iframe frameborder="0" id="frame3" name="frame3" allowfullscreen="true">

<button type="submit" form="nameform" value="Submit">Submit3</button>

</iframe>

</body>

</html>

**Selenium code :**

chrome.switchTo().frame("iframe1");

chrome.findElement(By.id("loginbutton")).click();

chrome.switchTo().defaultContent();

chrome.switchTo().frame("iframe2");

chrome.findElement(By.id("loginbutton")).click();

chrome.switchTo().defaultContent();

chrome.switchTo().frame("iframe3");

chrome.findElement(By.id("loginbutton")).click();

chrome.switchTo().defaultContent();

**Nested Frames:**



<html>

<head> </head>

<body class=" darktheme">

<iframe frameborder="0" id="iframe2" name=" iframe2" allowfullscreen="true">

<button type="submit" form="nameform" id=" element2" value="Submit">element2</button>

<iframe frameborder="0" id="iframe1" name=" iframe1" allowfullscreen="true">

<button type="submit" form="nameform" " id=" element1" value="Submit">element1</button>

</iframe>

</iframe>

</body>

</html>

**Selenium syntax :**

chrome.switchTo().frame("iframe2");

chrome.findElement(By.id("element2")).click();

chrome.switchTo().frame("iframe1");

chrome.findElement(By.id("element1")).click();

chrome.switchTo().parentFrame();

chrome.switchTo().defaultContent();

**Select Class in selenium**:

Select is used to automate dropdown which are developed by using <select> HTML tag.

Graphical user interface, text, application

Description automatically generated

**Types of dropdowns**:

1.single selection dropdown

Example :



2. Multiple selection dropdown

Example :



**How to check whether dropdown is multi-Select?**

The Select class provides the "**isMultiple()** " method, which determines whether the web element is supports for multiple selections or not. It returns a boolean value, i.e., True/False, without taking any argument. It checks the attribute 'multiple' in the HTML code for the web element.

Syntax :

Select country = new Select(driver.findElement(By.name("xxxxxx")));

Boolean selection = country.isMultiple();

System.out.println(selection);

**Select-Methods:**

1. **selectByIndex()** :

This method selects the dropdown option by its index number. We provide an integer value as the index number as an argument.

The index starts at 0

**syntax**:

Select country = new Select(driver.findElement(By.name("xxxxxx")));

country.selectByIndex(3);

**2.** **selectByValue() :**

This method selects the dropdown option by its value. We provide a string value as the value as an argument.

**syntax**:

Select country = new Select(driver.findElement(By.name("xxxxxx")));

country.selectByValue("india");

3. **selectByVisibleText() :**

This method selects the dropdown option based on the dropdown text. We provide the dropdown text as a string as an argument.

**Syntax**:

Select country = new Select(driver.findElement(By.name("xxxxxx")));

country.selectByVisibleText("INDIA");

4. **getOptions()** :

This method is used to get all the options in a dropdown or multi-select box.

Using this method, we can retrieve all the options of a dropdown (be it single-select or multi-select ).

This method returns all the options of the dropdown as a list of WebElement (List<WebElement>).

**syntax**:

Select country = new Select(driver.findElement(By.name("xxxxxx")));

List<WebElement>options = country.getOptions();

for(WebElement option:options)

{

System.out.println(option.getText());

}

5. **getFirstSelectedOption() :**

This method returns the first selected option of the dropdown. If it is a single-select dropdown, this method will return the selected value of the dropdown, and if it is a multi-select dropdown, this method will return the first selected value of the dropdown.

This method returns a WebElement.

**Syntax:**

Select country = new Select(driver.findElement(By.name("xxxxxx")));

System.out.println(country.getFirstSelectedOption());

6. **getAllSelectedOptions() :**

This method returns all the selected options of the dropdown. If it is a single-select dropdown, this method will return the only selected value of the dropdown, and if it is a multi-select dropdown, this method will return all the selected values of the dropdown.

Using this method, we can retrieve all the selected options of a dropdown (be it single-select or multi-select ).

This method returns a WebElement’s.

**Syntax:**

Select country = new Select(driver.findElement(By.name("xxxxxx")));

List<WebElement>options = country. getAllSelectedOptions ();

for(WebElement option:options)

{

System.out.println(option.getText());

}

7. **deselectAll():**

It will deselect all the options from the dropdown.

**Syntax:**

Select country = new Select(driver.findElement(By.name("xxxxxx")));

country.deselectAll()

8. **deselectByIndex ():**

This method is used to deselect an option from the dropdown by using its index.

**Syntax:**

Select country = new Select(driver.findElement(By.name("xxxxxx")));

country. deselectByIndex(index);

9. **deselectByValue ():**

This method is used to deselect an option from the dropdown by using its value.

**Syntax:**

Select country = new Select(driver.findElement(By.name("xxxxxx")));

country.deselectByValue(“Value”);

10. **deselectByVisibleText():**

This method is used to deselect an option from the dropdown by using its Text.

**Syntax:**

Select country = new Select(driver.findElement(By.name("xxxxxx")));

country.deselectByVisibleText(“Text”);

**Actions Class in selenium:**

**Actions Class:**

* Actions class is provided by selenium to perform mouse events and handling keyboard operations.
* Actions classes are used to perform multiple actions at a time sequentially.
* **build()** method is always the final method, so that all the listed actions will be compiled into a single step
* **perform()** method is used to perform all compiled/stored actions.

**Different Methods in Actions class to perform Keyboard Events:**

1. **keyDown(Keys):**

This method used to perform a key press(Keyboard).

1. **keyDown(WebElement** , **Keys):**

This method performs, cursor focus on given webelement and key press(Keyboard).

1. **keyUp(Keys):**

This method used to perform a key release(Keyboard).

1. **keyUp(WebElement** , **Keys):**

This method performs, cursor focus on given webelement and key release(Keyboard).

1. **sendKeys(keys):**

This method used to perform Sendkeys to the active web element.

1. **sendKeys(WebElement** , **keys):**

This method performs, cursor focus on given webelement and Sendkeys.

Example for keyDown, KeyUp & sendKeys :

**Syntax:**

Actions action = new Actions(chrome);

WebElement firstname = chrome.findElement(By.name("firstName"));

action.click(firstname).keyDown(firstname,Keys.SHIFT)

.sendKeys(firstname, "siddhiInstitute")

.keyUp(firstname,Keys.SHIFT).build().perform();

**Different Methods in Actions class to perform Mouse Events:**

1. **moveToElement(WebElement):** This method used to move the mouse pointer to the middle of the WebElement.

**Syntax:**

Actions action = new Actions(chrome);

WebElement nextbutton = chrome.findElement(By.name("xxxxxx"));

action.moveToElement(nextbutton).build().perform();

1. **click():** This method used to click at the **current** mouse pointer.

**Syntax:**

Actions action = new Actions(chrome);

action.click().build().perform();

1. **click(WebElement):** This method used to perform click on the WebElement, which is passed an argument.

**Syntax:**

Actions action = new Actions(chrome);

WebElement nextbutton = chrome.findElement(By.name("xxxxxx"));

action.click(nextbutton).build().perform();

1. **doubleClick():** This method used to double-click at the current mouse pointer.

**Syntax:**

Actions action = new Actions(chrome);

action.doubleClick().build().perform();

1. **doubleClick(WebElement):** This method used to perform double-click on the WebElement, which is passed an argument.

**Syntax:**

Actions action = new Actions(chrome);

WebElement nextbutton = chrome.findElement(By.name("xxxxxx"));

action.doubleClick(nextbutton).build().perform();

1. **contextClick()** : This method used to perform a context-click(mouse right click) at current mouse pointer.

**Syntax:**

Actions action = new Actions(chrome);

action.contextClick().build().perform();

1. **contextClick(WebElement)** : This method used to perform a context-click(mouse right click) at middle of the given WebElement.

**Syntax:**

Actions action = new Actions(chrome);

WebElement menu = chrome.findElement(By.name("xxxxxx"));

action.contextClick(menu).build().perform();

1. **clickAndHold():** This method used to perform click (without releasing) at current mouse pointer.

**Syntax:**

Actions action = new Actions(chrome);

action.clickAndHold().build().perform();

1. **clickAndHold(WebElement):** This method used to perform clicks (without releasing) in the middle of the given WebElement.

**Syntax:**

Actions action = new Actions(chrome);

WebElement imagetohold = chrome.findElement(By.name("xxxxxx"));

action.clickAndHold(imagetohold).build().perform();

1. **release():** This method used to release the depressed left mouse button at the current mouse location.

**Syntax:**

Actions action = new Actions(chrome);

action.release().build().perform();

1. **release(WebElement):** This method used to release the depressed left mouse button on the webelement.

**Syntax:**

Actions action = new Actions(chrome);

WebElement imagetohold = chrome.findElement(By.name("xxxxxx"));

action.clickAndHold(imagetohold).build().perform();

action.release(imagetohold).build().perform();

1. **dragAndDrop(source, target):** This method used to perform click-and-hold at the location of the source webelement, moves and release at the location of the target webelement.

**Syntax**:

WebElement source = chrome.findElement(By.xpath(“xxx"));

WebElement target = chrome.findElement(By.xpath(“xxx"));

action.dragAndDrop(source, target).build().perform();

1. **dragAndDropBy(source, xOffset, yOffset):** This method used to perform click-and-hold the mouse at the location of the source webelement, moves and release at given offset.

**Syntax**:

WebElement source = chrome.findElement(By.xpath(“xxx"));

action.dragAndDrop(source, x, y).build().perform();

1. **moveByOffset(x-offset, y-offset):** This method used to move the mouse from its current position (or 0,0) by the given offset.

**Syntax**:

WebElement source = chrome.findElement(By.xpath(“xxx"));

action.moveByOffset( x, y).build().perform();

1. **pause(long):** This method is used to pause/delay the execution between actions for a specified duration in milli seconds

**Syntax**:

Actions action = new Actions(chrome);

action.click().pause(2000).click().build().perform();

1. **pause(Duration ):** This method is used to pause/delay the execution between actions for a specified duration.

**Syntax**:

Actions action = new Actions(chrome);

action.click(firstname).pause(Duration.ofSeconds(2)).sendKeys("xx”).build().perform();

**JavaScriptExecuter in selenium:**

**What is JavaScriptExecuter ?**

* Javascriptexecutor is an interface which is used to execute JavaScript with selenium webdriver.
* Javascriptexecutor allows you to run pure JavaScript code irrespective of the Selenium language binding(Java, C#, Python etc.) you are using.

**Why JavaScriptExecuter?**

* JavaScript is extremely efficient and accurate when comparing with scripts written with Selenium-Java/c#/python etc.
* some features we can’t handle using Java-selenium so we need scripting language also which can control server side or client-side scripting so we will use JavaScript in our Selenium

It consists of two methods used to run JavaScript

1. executeScript : This method is used to execute JavaScript in sequential order.
2. executeAsyncScript : This method is used to execute asynchronous script in random order.(Time constants)

**Note** : Both methods will block the WebDriver execution until they complete execution

**JavaScriptExecuter Actions:**

1. **Navigating URL :**

JavascriptExecutor js = (JavascriptExecutor) driver; js.executeScript("window.location=’url’");

1. **Title of a Webpage :**

JavascriptExecutor js = (JavascriptExecutor) driver;

String title = js.executeScript("return document.title").toString();

System.***out***.println(title);

1. **Domain of a Webpage :**

JavascriptExecutor js = (JavascriptExecutor) driver;

String domain = js.executeScript("return document.domain").toString();

System.***out***.println(domain);

1. **URL of a Webpage**

JavascriptExecutor js = (JavascriptExecutor) driver;

String url = js.executeScript("return document.URL").toString();

System.***out***.println(url);

1. **Inner Text of a WebElement :**

JavascriptExecutor js = (JavascriptExecutor) driver;

String text = js.executeScript("return arguments[0].innerText",WebElement).toString();

System.***out***.println(text);

(or)

String text1 = js.executeScript("return document.getElementById('xx').innerText").toString();

System.***out***.println(text1);

1. **Attribute of a WebElement:**

JavascriptExecutor js = (JavascriptExecutor) driver;

String arg = js.executeScript("return arguments[0].getAttribute('xxx')",email).toString();

System.***out***.println(arg);

(or)

String arg1 = js.executeScript("return document.getElementById('xxx').getAttribute('xxx')").toString();

System.***out***.println(arg1);

1. **click on WebElement**

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("arguments[0].click()",WebElement);

(or)

js.executeScript("document.getElementById('xxx').click();");

1. **sendkeys on WebElement**

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("arguments[0].value='xxxx'", WebElement);

(or)

js.executeScript("document.getElementById('xx').value='xxx'");

1. **vertical scroll**

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("window.scrollTo(0,300)");

1. **horizontal scroll**

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("window.scrollTo(300,0)");

1. **scroll bottom of the webpage**

JavascriptExecutor js = (JavascriptExecutor) driver;

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

1. **scroll top of the webpage**

JavascriptExecutor js = (JavascriptExecutor) driver;

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

1. **Height of a webpage**

JavascriptExecutor js = (JavascriptExecutor) driver;

String height = js.executeScript("return window.innerHeight").toString();

System.***out***.println(height);

1. **Width of a webpage**

JavascriptExecutor js = (JavascriptExecutor) driver;

String width = js.executeScript("return window.innerWidth").toString();

System.***out***.println(width);

1. **changing the border color**

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("arguments[0].style.borderColor='yellow'", WebElement);

1. **scroll Into View**

JavascriptExecutor js = (JavascriptExecutor) driver; js.executeScript("document.getElementById('xx').scrollIntoView(true)"); (or)

js.executeScript("arguments[0].scrollIntoView(true);", WebElement);

1. **multiple actions**

JavascriptExecutor js = (JavascriptExecutor) driver; js.executeScript("arguments[0].click();arguments[1].click();",

WebElement1, WebElement2);

1. **Example for asyncscript**

JavascriptExecutor js = (JavascriptExecutor) driver; js.executeAsyncScript("window.setTimeout(arguments[arguments.length-1],5000);alert('Hello');");

1. **alert creation**

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("alert('Hello');");

1. **Refresh webpage**

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("history.go(0)");

**Screenshot in Selenium Webdriver :**

**TakesScreenShot** is an interface which is used to take screen of a web page during the execution for debugging the failures.

To take screen shot first we need to convert webdriver instance into TakeScreenShot.

Jar files required to add project : **Apache Commons IO**

**TakesScreenshot Methods :**

1. **getScreehshotAs(OutputType) :** This method used to take screen shot and convert into image file.

**Syntax :**

TakesScreenshot screenshot = (TakesScreenshot) chrome;

File source = screenshot.getScreenshot(OutputType.***FILE***);

File dest = **new** File("path where to store");

FileUtils.*copyFile*(source, dest);

**FileUpload in selenium:**

We can use webdriver sendKeys method to upload file in file upload option. But that file upload WebElement must have **<input> tag** and attribute **type=’file’**.

Example of file upload html :

Text

Description automatically generated

**Syntax :**

chrome.findElement(By.*id*("xxxx")).sendKeys("path");

**FileUpload with Robot Class:**

Robot class used to operate all keyboard event.

chrome.findElement(By.*id*("uploadbutton")).click();

StringSelection path = **new** StringSelection("C:\\xxxxx\\xxxx.png");

Toolkit.*getDefaultToolkit*().getSystemClipboard().setContents(path,**null**);

Robot robot = **new** Robot();

robot.keyPress(KeyEvent.***VK\_ENTER***);

robot.keyRelease(KeyEvent.***VK\_ENTER***);

robot.keyPress(KeyEvent.***VK\_CONTROL***);

robot.keyPress(KeyEvent.***VK\_V***);

robot.keyRelease(KeyEvent.***VK\_V***);

robot.keyRelease(KeyEvent.***VK\_CONTROL***);

robot.keyPress(KeyEvent.***VK\_ENTER***);

robot.keyRelease(KeyEvent.***VK\_ENTER***);

**FileUpload with Auto IT:**

**Software required to download :**

1. **AutoIT :** To read window information.
2. **Script Editor :** To write autoit script.

**Syntax:**

chrome.findElement(By.*id*("uploadbutton")).click();

ControlFocus("Open","","Edit1")

ControlSetText("Open","","Edit1","path of file to upload.png")

ControlClick("Open","","Button1")

Runtime.*getRuntime*().exec("path of compiled autoit script.exe");

**Keys class in selenium:**

Keys class used to perform complete keyboard actions.

Syntax:

chrome.findElement(By.*xpath*("xx")).sendKeys(Keys.***CONTROL***);

**Xpath in selenium:**