Mouse Click & Keyboard Event: Action Class in Selenium Webdriver

Action Class in Selenium

Action Class in Selenium is a built-in feature provided by the selenium for handling keyboard and mouse events. It includes various operations such as multiple events clicking by control key, drag and drop events and many more. These operations from the action class are performed using the advanced user interaction API in Selenium Webdriver.

Handling Keyboard & Mouse Events

Handling special keyboard and mouse events are done using the Advanced User Interactions API. It contains the Actions and the Action classes that are needed when executing these events. The following are the most commonly used keyboard and mouse events provided by the Actions class.

Step 1: Import the Actions and Action classes.

Step 2: Instantiate a new Actions object.

Step 3: Instantiate an Action using the Actions object in step 2.

In this case, we are going to use the moveToElement() method because we are simply going to mouse-over the "Home" link.

Step 4: Use the perform() method when executing the Action object we designed in Step 3.

MouseHover:

Test url: “https://opensource-demo.orangehrmlive.com/index.php/auth/validateCredentials”

package newproject;

import org.openqa.selenium.\*;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.interactions.Action;

import org.openqa.selenium.interactions.Actions;

public class PG7 {

public static void main(String[] args) {

String baseUrl = "http://demo.guru99.com/test/newtours/";

System.setProperty("webdriver.gecko.driver","C:\\geckodriver.exe");

WebDriver driver = new FirefoxDriver();

driver.get(baseUrl);

WebElement link\_Home = driver.findElement(By.linkText("Home"));

WebElement td\_Home = driver

.findElement(By

.xpath("//html/body/div"

+ "/table/tbody/tr/td"

+ "/table/tbody/tr/td"

+ "/table/tbody/tr/td"

+ "/table/tbody/tr"));

Actions builder = new Actions(driver);

Action mouseOverHome = builder

.moveToElement(link\_Home)

.build();

String bgColor = td\_Home.getCssValue("background-color");

System.out.println("Before hover: " + bgColor);

mouseOverHome.perform();

bgColor = td\_Home.getCssValue("background-color");

System.out.println("After hover: " + bgColor);

driver.close();

}

}

Building a Series of Multiple Actions

You can build a series of actions using the Action and Actions classes. Just remember to close the series with the build() method. Consider the sample code below.

public static void main(String[] args) {

String baseUrl = "http://www.facebook.com/";

WebDriver driver = new FirefoxDriver();

driver.get(baseUrl);

WebElement txtUsername = driver.findElement(By.id("email"));

Actions builder = new Actions(driver);

Action seriesOfActions = builder

.moveToElement(txtUsername)

.click()

.sendKeys(txtUsername, "hello")

.doubleClick(txtUsername)

.contextClick()

.build();

seriesOfActions.perform() ;

}

# PopUps and Alerts in Selenium

A lot of the times, when we are filling in a form and press the submit button, an alert pops up, telling me, “Contact Number is Required. ” At times, I forget to check the terms and conditions box, and I get the same as a notification via a popup-up alert. Pressing the ***OK***  button on the alert box takes me back to the form, and the form does not submit until I have filled everything “required. ” These use cases show how necessary are the “Alerts” or “Popups“ in Web Applications.

## What are Alerts/popups in Selenium?

***Alerts*** are small popup boxes/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.

***For Example,*** The alert box displayed below requires an action from the user to press ***OK*** and accept or press ***Cancel*** and dismiss the message box.

### ***What are the different types of Alerts/popups?***

1) Simple Alert

The simple alert class in Selenium displays some information or warning on the screen.

2) Prompt Alert.

This Prompt Alert asks some input from the user and Selenium webdriver can enter the text using sendkeys(" input…. ").

3) Confirmation Alert.

These alerts ***get some confirmation*** from the user in the form of ***accepting*** or ***dismissing*** the message box.

**How to handle Alert in Selenium WebDriver**

Alert interface provides the below few methods which are widely used in Selenium Webdriver.

1) void dismiss() // To click on the 'Cancel' button of the alert.

driver.switchTo().alert().dismiss();

2) void accept() // To click on the 'OK' button of the alert.

driver.switchTo().alert().accept();

3) String getText() // To capture the alert message.

driver.switchTo().alert().getText();

4) void sendKeys(String stringToSend) // To send some data to alert box.

driver.switchTo().alert().sendKeys("Text");

Now we automate the given below scenario.

In this scenario, we will use Guru99 demo site to illustrate Selenium Alert handling.

Step 1) Launch the web browser and open the site "http://demo.guru99.com/test/delete\_customer.php "

Step 2) Enter Any Customer id.

Step 3) After entering the customer ID, Click on the "Submit" button.

Step 4) Reject/accept the alert.

Handling Alert in Selenium Webdriver using above scenario

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.NoAlertPresentException;

import org.openqa.selenium.Alert;

public class AlertDemo {

public static void main(String[] args) throws NoAlertPresentException,InterruptedException {

System.setProperty("webdriver.chrome.driver","G:\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

// Alert Message handling

driver.get("http://demo.guru99.com/test/delete\_customer.php");

driver.findElement(By.name("cusid")).sendKeys("53920");

driver.findElement(By.name("submit")).submit();

// Switching to Alert

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

// Capturing alert message.

String alertMessage= driver.switchTo().alert().getText();

// Displaying alert message

System.out.println(alertMessage);

Thread.sleep(5000);

// Accepting alert

alert.accept();

}

}

**What is Drag and Drop Action?**

This is an action performed with a ***mouse***when a user moves ***(drags)*** a web element and then places **(*drops*)** it into an alternate area.

* *Select some element on the web page, drag it and then place it on the alternate area.*

To perform the drag-drop action through a Selenium script, there is no direct drag-drop method available in*WebElement interface*. Unlike other commands like *click()*, *sendKeys()* there is nothing available for drag and drop. Here, we leverage the [***Actions class***](https://www.toolsqa.com/selenium-webdriver/actions-class-in-selenium/) which provides various methods of emulating such complex interactions.

So, here are the methods Actions class provides for Drag-Drop action:

***dragAndDrop(WebElementsource,****WebElement target)*

## Drag and Drop in Selenium

***dragAndDrop(WebElement source, WebElement target):*** This method performs left click, hold the click to hold the source element, moves to the location of the target element and then releases the mouse click.

Let’s see how to use Action class methods to perform drag-drop action:

***First, instantiate an Actions class:***

    Actions actions = new Actions(driver);

As you can see, the *dragAndDrop(WebElement source, WebElement target)* method has two arguments to pass. One is a source web element and another is target web element.  This source web element is any web element that needs to be dragged. Target web element is any web element on which dragged object needs to be placed or dropped. To find the source and target element use the below command:

*WebElement source = driver.findElement(Any By strategy & locator);*

*WebElement target = driver.findElement(Any By strategy & locator);*

Now, when we have got the actions class object and the element as well, just invoke***perform()***method for the drag & drop:

*actions.dragAndDrop(source,target).perform();*

Let’s see what happens internally when invoke the *perform()*method above:

* ***Click And Hold Action****: dragAndDrop() method first performs click-and-hold at the location of the source element*

***clickAndHold(sourceElement)***;

* ***Move Mouse Action****: Then source element gets moved to the location of the target element*

***moveToElement(targetElement)***;

* ***Button Release Action****: Finally, it releases the mouse*

***release()***;

* ***Build****: build() method is used to generate a composite action containing all actions. But if you observe, we have not invoked it in our above command. The build is executed in the perform method internally*
* ***Perform****: perform() method performs the actions we have specified. But before that, it internally invokes build() method first. After the build, the action is performed*

**Another way:** use url to test: ” <http://www.dhtmlgoodies.com/scripts/drag-drop-custom/demo-drag-drop-3.html>”

Identify that both elements

WebElement source=driver.findElement(By.*xpath*("//div[@id='box1']"));

WebElement target=driver.findElement(By.*xpath*("//div[@id='box103']"));

Actions ac=new Actions(driver);

ac.clickAndHold(source).moveToElement(target).release().perform();

# **How to handle multiple windows in Selenium?**

While automating any website or a web application, we must have witnessed a scenario where multiple windows open within an application when a button is clicked and the user has to perform some action on all the opened windows. Now user might not be able to work on all windows at the same time and hence need some mechanism through which he can take control over parent and child windows or if I may say from a QA’s perspective, how can we handle window using Selenium?

## What is a window in Selenium?

**A window in any browser is the main webpage on which the user is landed after hitting a link/URL.**Such a window in [**Selenium**](https://www.toolsqa.com/selenium-webdriver/selenium-testing/) is referred to as the ***parent window*** also known as the ***main window*** which opens when the Selenium WebDriver session is created and has all the focus of the WebDriver.

### How do we identify parent window and child windows?

When a user hits a URL, a webpage opens. This main page is the ***parent window***  i.e the main window on which the user has currently landed and will perform any operation. This is the same webpage that will open when our Selenium automation script will execute. All the windows which will open inside your main window will be termed as ***child windows***.

## What is a window handle in Selenium?

**A window handle stores the unique address of the browser windows.** It is just a pointer to a window, whose return type is alphanumeric. The window handle in Selenium helps in handling multiple windows and child windows. Each browser will have a unique window handle value with which we can uniquely identify it.

***For example:*** When you open a website say ***” https://demoqa.com/browser-windows”*** and click on browser windows inside, each opened window will have an ID using which we will be able to switch the Selenium WebDriver context to that window and perform any operation within that window. In the image below you can see three windows – one parent window and two child windows. All three of them will have a unique window handle i.e an ID.

### ***What are the different methods used for window handling in Selenium?***

**Selenium WebDriver provides various methods for handling of windows. Few of them are:**

* ***getWindowHandle( ):***When a website opens, we need to handle the main window i.e the parent window using driver.getWindowHandle( ); method. With this method, we get a unique ID of the current window which will identify it within this driver instance. This method will return the value of the [***String type***](https://www.toolsqa.com/java/basic-java-programming/string-class/)***.***
* ***getWindowHandles( ):***To handle all opened windows which are the ***child windows*** by web driver, we use driver.getWindowHandles( ); method. The windows store in a ***Set*** of String type and here we can see the transition from one window to another window in a web application. Its return type is[***Set***](https://www.toolsqa.com/java/list-interface/#:~:text=Method%3A%20iterator()&text=For%20the%20reader%20unfamiliar%20with,point%20to%20the%20next%20page.&text=%2F%2F%20Get%20a%20list%20iterator%20over%20the%20elements%20in%20the%20list.,-ListIterator%20listIterator)***<String>.***
* ***switchto():*** Using this method we perform switch operation within windows.
* ***action***: This method helps in performing certain actions on the windows.

## How do we handle child windows in Selenium?

As seen in the above example if we have child windows in any web application then interaction with them without proper window handling will lead to an exception. For this we have different methods explained above, we will use them here with a practical example.

We will be using ***getWindowHandle( ) and getWindowHandles( )***method here along with ***switchto()*** method.

Taking the same example of [**” ToolsQA Demo Site“**](https://demoqa.com/browser-windows) above where exception was encountered, we will show how it will be executed successfully. After opening the URL, we will click on the “**New window”**button within the application**,**a new browser window opens. We will read the text from the newly opened window i.e ***” This is sample page”*** and will print it.

Launch the browser and open the site **https://demoqa.com/browser-windows** "

package com.java.testing;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.interactions.Action;

import org.openqa.selenium.interactions.Actions;

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

import java.util.Iterator;

import java.util.Set;

import org.openqa.selenium.\*;

public class Form {

public static void main(String[] args) throws InterruptedException {

System.setProperty("webdriver.gecko.driver",

"C:\\Users\\mkiran\\Desktop\\TestJava\\Testing\\Latest\\drivers\\geckodriver-v0.29.0-win64\\geckodriver.exe");

WebDriver driver = new FirefoxDriver();

driver.manage().window().maximize();

driver.get("https://demoqa.com/browser-windows");

// Open new child window within the main window

driver.findElement(By.id("windowButton")).click();

// Get handles of the windows

String mainWindowHandle = driver.getWindowHandle();

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

Iterator<String> iterator = allWindowHandles.iterator();

// Here we will check if child window has other child windows and will fetch the

// heading of the child window

while (iterator.hasNext()) {

String ChildWindow = iterator.next();

System.out.println("\*\*\*\*\*\*\* ChildWindow \*\*\*\*\*\*\*\*\*\*"+ChildWindow);

if (!mainWindowHandle.equalsIgnoreCase(ChildWindow)) {

driver.switchTo().window(ChildWindow);

WebElement text = driver.findElement(By.id("sampleHeading"));

System.out.println("Heading of child window is " + text.getText());

driver.close();

}

}

driver.switchTo().window(mainWindowHandle);

driver.close();

}

}

**How to Verify Tooltip using Selenium WebDriver**

**Tooltip in Selenium**

A Tooltip in Selenium is a text that appears when a mouse hovers over an object on a web page. The object can be a link, an image, a button, a text area, etc. The tooltip text often gives more information about the object on which the user hovers over the mouse cursor.

Tooltips were traditionally implemented as a 'title' attribute to an element. The value of this attribute was shown as a tooltip on mouse-hover. This is a static text giving information of the element with no styling.

**How to get value of Attribute:**

**getAttribute(“Attrubute name”)** //return the value of attribute

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.\*;

public class ToolTip {

public static void main(String[] args) {

String baseUrl = " https://omayo.blogspot.com/ ";

System.setProperty("webdriver.chrome.driver","G:\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get(baseUrl);

String expectedTooltip = "search";

// Find the Github icon at the top right of the header

WebElement github = driver.findElement(By.xpath("//tbody/tr[1]/td[2]/input[1] "));

//get the value of the "title" attribute of the search icon

String actualTooltip = github.getAttribute("title");

//Assert the tooltip's value is as expected

System.out.println("Actual Title of Tool Tip"+actualTooltip);

if(actualTooltip.equals(expectedTooltip)) {

System.out.println("Test Case Passed");

}

driver.close();

}

}