Actions

Selenium has the built-in ability to handle various types of keyboard and mouse events. In this post, we’ll teach you about the Selenium Actions class which enables user interaction with the web applications.

To perform mouse operations on a website, first of all, we’ve to launch the browser and then navigate to the URL of the site. The entire control of the browser as well as the application is in the reference variable of type WebDriver.

The WebDriver reference variable can identify any web element which is present on the page. But it doesn’t have the ability to handle the following mouse events.

* **Mouse actions on some elements.**
* **Drag and drop.**
* **Right-click, and so on.**

Hence, you should clearly know that the WebDriver reference variable alone can’t work. So, to perform the mouse actions, we need to make use of the Selenium Actions class.

## Selenium Actions Class – Handle Keyboard And Mouse Events

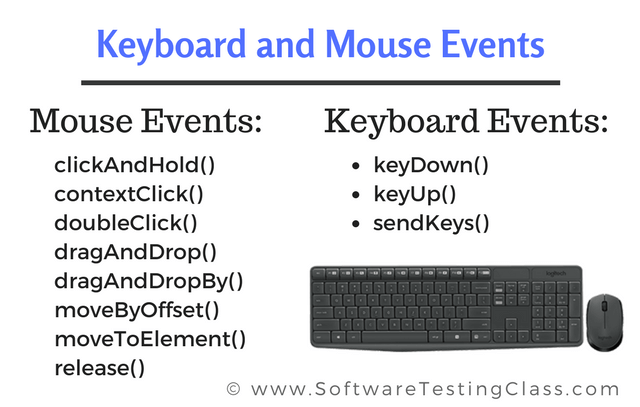
WebDriver driver = new FirefoxDriver();

WebElement ele = driver.findElement(By.xpath(…………));

Actions ref = new Actions (driver);

ref.movetoElement(ele);

ref.click().build().perform();



### Some common used Actions Class Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| **<clickAndHold()>** | Clicks at the present mouse location (without releasing). |
| **<contextClick()>** | Makes a context-click at the existing mouse location. |
| **<doubleClick()>** | It performs a double-click at the existing mouse location. |
| **<dragAndDrop(source, target)>** | Invokes click-and-hold at the source location and moves to the location of the target element before releasing the mouse.  **Parameters:**  source– element to grab.  target– element to release. |
| **<dragAndDropBy(source, x-offset, y-offset)>** | Performs click-and-hold at the source location, shifts by a given offset, then frees the mouse.  **Parameters**:  source- element to grab.  <xOffset>- to shift horizontally.  <yOffset>- to shift vertically. |
| **<keyDown(modifier\_key)>** | Invokes a modifier key press. Does not free the modifier key – subsequent actions may assume it as pressed.  **Parameters**:  modifier\_key – any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| **<keyUp(modifier \_key)>** | Frees up a mouse press.  **Parameters**:  modifier\_key – any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| **<moveByOffset(x-offset, y-offset)>** | Shifts the mouse from its current position (or 0,0) by the given offset.  **Parameters**:  <x-offset> – Sets the horizontal offset. A (-ve) value means shifting the mouse to the left.  <y-offset> – Sets the vertical offset. A (-ve) value means shifting the mouse to the up. |
| **<moveToElement(toElement)>** | It shifts the mouse to the center of the element.  **Parameters:**  <toElement> – target element. |
| **<release()>** | Frees up the depressed left mouse button at the existing mouse location. |
| **<sendKeys(onElement, charsequence)>** | Transmits a series of keystrokes onto the element.  **Parameters:**  <onElement> – an element that will receive the keystrokes, usually a text field.  <charsequence> – any string value representing the sequence of keystrokes to be sent. |