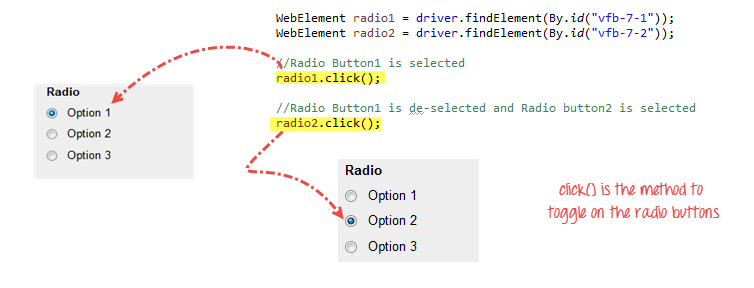
1. Selenium Form WebElement: TextBox, Submit Button, sendkeys(), click()  
   Forms are the fundamental web elements to receive information from the website visitors. Web forms have different GUI elements like Text boxes, Password fields, Checkboxes, Radio buttons, dropdowns, file inputs, etc.

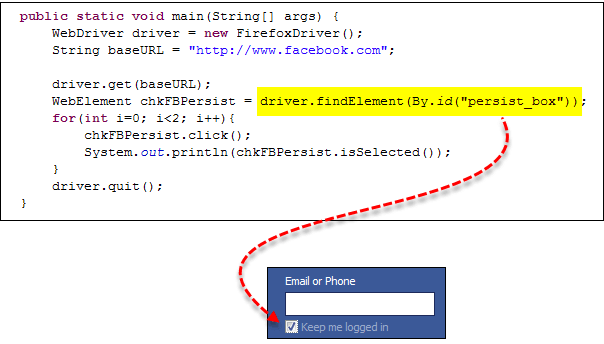
**Selenium encapsulates every form element as an object of WebElement.** It provides API to find the elements and take action on them like entering text into text boxes, clicking the buttons, etc.

2.How to Select Check Box and Radio Button in Selenium WebDriver

Both Check Box and Radio Button uses click() method to toggle. The radio button will toggle between multi options, check box will toggle between on/off.

Radio Button Eg: 

Check Box Eg:



3.How to Click on Image in Selenium Webdriver

Accessing Image Links:

We cannot use the By.linkText() and By.partialLinkText() methods because image links basically have no link texts at all.

In this case, we should resort to using either By.cssSelector or By.xpath. The first method is more preferred because of its simplicity.

Eg: driver.findElement(By.cssSelector("a[title=\"Go to Facebook home\"]")).click();

4.How to Select Value from DropDown using Selenium Webdriver (Reffer: Selenium Day3)

5.Mouse Click & Keyboard Event: Action Class 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.

|  |  |
| --- | --- |
| Method | Description |
| clickAndHold() | Clicks (without releasing) at the current mouse location. |
| contextClick() | Performs a context-click at the current mouse location. (Right Click Mouse Action) |
| doubleClick() | Performs a double-click at the current mouse location. |
| dragAndDrop(source, target) | Performs click-and-hold at the location of the source element, moves to the location of the target element, then releases the mouse.  Parameters:  source- element to emulate button down at.  target- element to move to and release the mouse at. |
| dragAndDropBy(source, x-offset, y-offset) | Performs click-and-hold at the location of the source element, moves by a given offset, then releases the mouse.  Parameters:  source- element to emulate button down at.  xOffset- horizontal move offset.  yOffset- vertical move offset. |
| keyDown(modifier\_key) | Performs a modifier key press. Does not release the modifier key - subsequent interactions may assume it's kept pressed.  Parameters:  modifier\_key - any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| keyUp(modifier \_key) | Performs a key release.  Parameters:  modifier\_key - any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| moveByOffset(x-offset, y-offset) | Moves the mouse from its current position (or 0,0) by the given offset.  Parameters:  x-offset- horizontal offset. A negative value means moving the mouse left.  y-offset- vertical offset. A negative value means moving the mouse down. |
| moveToElement(toElement) | Moves the mouse to the middle of the element.  Parameters:  toElement- element to move to. |
| release() | Releases the depressed left mouse button at the current mouse location |
| sendKeys(onElement, charsequence) | Sends a series of keystrokes onto the element.  Parameters:  onElement - element that will receive the keystrokes, usually a text field  charsequence - any string value representing the sequence of keystrokes to be sent |

**Step 1:**Import the **Actions** and **Action** classes.

[Keyboard & Mouse Event using Action Class in Selenium Webdriver](https://www.guru99.com/images/image047.png)

**Step 2:**Instantiate a new Actions object.

[Keyboard & Mouse Event using Action Class in Selenium Webdriver](https://www.guru99.com/images/image048.png)

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

[Keyboard & Mouse Event using Action Class in Selenium Webdriver](https://www.guru99.com/images/image049.png)

In this case, we are going to use the moveToElement() method because we are simply going to mouse-over the "Home" link. The build() method is always the final method used so that all the listed actions will be compiled into a single step.

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

[Keyboard & Mouse Event using Action Class in Selenium Webdriver](https://www.guru99.com/images/image050.png)

6.How to Upload & Download a File using Selenium Webdriver

**Upload Files:**

Uploading files in WebDriver is done by simply using the sendKeys() method on the file-select input field to enter the path to the file to be uploaded.

driver.get(baseUrl);

WebElement uploadElement = driver.findElement(By.id("uploadfile\_0"));

// enter the file path onto the file-selection input field

uploadElement.sendKeys("C:\\newhtml.html");

// check the "I accept the terms of service" check box

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

// click the "UploadFile" button

driver.findElement(By.name("send")).click();

Remember following two things when uploading files in WebDriver

1. There is no need to simulate the clicking of the "Browse" button. WebDriver automatically enters the file path onto the file-selection text box of the <input type="file"> element
2. When setting the file path in your Java IDE, use the proper escape character for the back-slash.

### **Downloading Files:**

**WebDriver has no capability to access the Download dialog boxes**presented by browsers when you click on a download link or button. However, we can bypass these dialog boxes using a separate program called "wget".

#### What is Wget?

**Wget is a small and easy-to-use command-line program used to automate downloads**. Basically, we will access Wget from our WebDriver script to perform the download process.

#### Setting up Wget

**Step 1:**In your C Drive, create a new folder and name it as "Wget".

Download wget.exe [from here](https://eternallybored.org/misc/wget/)(<https://eternallybored.org/misc/wget/>) and Place it in the Wget folder you created from the step above.

**Step 2:**Open Run by pressing windows key + "R" ; type in "cmd & click ok

Type in the command "cd /" to move to the root directory

**Step 3:**Type in the command to check whether the given setup is working

cmd /c C:\\Wget\\wget.exe -P C: --no-check-certificate http://demo.guru99.com/selenium/msgr11us.exe

There seems to be an issue writing into C drive.

**Step 4:**You need to debug the wget errors in command line before you execute the code using Selenium Webdriver. These errors will persist in Eclipse and the error messages will not be as informative. Best to first get wget working using command line. If it works in command line it will definitely work in Eclipse.

In our example, as show in step 3, there is a problem writing into C drive. Let's change the download location to D drive and check results.

cmd /c C:\\Wget\\wget.exe -P D: --no-check-certificate http://demo.guru99.com/selenium/msgr11us.exe

Messenger was downloaded successfully.

Before you proceed further don't forget to delete the downloaded file

### **Using WebDriver and Wget:**

**Step 1**

Import the "java.io.IOException" package because we will have to catch an IOException later in Step 4.

import java.io.IOException;

**Step 2**

Use getAttribute() to obtain the "href" value of the download link and save it as a String variable. In this case, we named the variable as "sourceLocation".

WebElement downloadButton = driver.findElement(By

.id("messenger-download"));

String sourceLocation = downloadButton.getAttribute("href");

**Step 3**

Set-up the syntax for wget using the following command.

String wget\_command = "cmd /c C:\\Wget\\wget.exe -P D: --no-check-certificate " + sourceLocation;

**Step 4**

Initiate the download process by calling wget from our WebDriver code.

try {

Process exec = Runtime.getRuntime().exec(wget\_command);

int exitVal = exec.waitFor();

System.out.println("Exit value: " + exitVal);

} catch (InterruptedException | IOException ex) {

System.out.println(ex.toString());

}

7.Robot Class in Selenium Webdriver

In certain Selenium Automation Tests, there is a need to control keyboard or mouse to interact with OS windows like Download pop-up, Alerts, Print Pop-ups, etc. or native Operation System applications like Notepad, Skype, Calculator, etc.

Selenium Webdriver cannot handle these OS pop-ups/applications.

In[Java](https://www.guru99.com/java-tutorial.html)version 1.3 Robot Class was introduced. Robot Class can handle OS pop-ups/applications.8.Handling iFrames in Selenium Webdriver: switchTo()

## Benefits of Robot Class

1. Robot Class can simulate Keyboard and Mouse Event
2. Robot Class can help in upload/download of files when using selenium web driver
3. Robot Class can easily be integrated with current automation framework (keyword, data-driven or hybrid)

9.Implicit, Explicit, & Fluent Wait in Selenium WebDriver(Refer: Selenium Day4)

10.Double click and Right Click in Selenium with Examples

**Double click:**

doubleClick() - Performs a double-click at the current mouse location.

Eg: Actions actions = new Actions(driver);

WebElement elementLocator = driver.findElement(By.id("ID"));

actions.doubleClick(elementLocator).perform();

**Right Click:**

contextClick() - Performs a context-click at the current mouse location. (Right Click Mouse Action)

Eg: Actions actions = new Actions(driver);

WebElement elementLocator = driver.findElement(By.id("ID"));

actions.contextClick(elementLocator).perform();