**Selenium Tutorial**

Contents

[Web Elements 2](#_Toc98187802)

[Locators 3](#_Toc98187803)

[Xpath 3](#_Toc98187804)

[CSS Selector 5](#_Toc98187805)

[Wait (Test / Page Synchronization) 6](#_Toc98187806)

[Visibility of WebElement 9](#_Toc98187807)

[Window Handles 9](#_Toc98187808)

[Browser Navigation Commands 10](#_Toc98187809)

[Actions Class (Keyboard and Mouse activities) 10](#_Toc98187810)

[Alerts Windows 13](#_Toc98187811)

[JavaScriptExecuter 14](#_Toc98187812)

[Iframes 15](#_Toc98187813)

[Model Window 16](#_Toc98187814)

[Turn off notifications 16](#_Toc98187815)

[Take Screen Shot 16](#_Toc98187816)

[Advanced Selenium 17](#_Toc98187817)

[AutoIT 17](#_Toc98187818)

[Robot Class 17](#_Toc98187819)

# Web Elements

List of very important Elements:

1. Textbox
2. Button
3. Anchor Link – a href
4. Check Box
5. Radio button
6. Dropdown box / Select
7. Date Picker
8. Text area
9. Image, Image link and Image button

List of few HTML Web Elements:

<https://www.w3schools.com/tags/default.asp>

* <input type="text"> (default value)
* <input type="button">
* <input type="checkbox">
* <input type="color">
* <input type="date">
* <input type="datetime-local">
* <input type="email">
* <input type="file">
* <input type="hidden">
* <input type="image">
* <input type="month">
* <input type="number">
* <input type="password">
* <input type="radio">
* <input type="range">
* <input type="reset">
* <input type="search">
* <input type="submit">
* <input type="tel">
* <input type="time">
* <input type="url">
* <input type="week">

Ex:

<http://register.rediff.com/register/register.php?FormName=user_details>

<https://ssc.nic.in/Registration/Home>

<https://www.jeevansathi.com/register/customreg/15?source=d_b_genlp1&sh1=Meet%20over&sh2=10%20lakh%20profiles&campaignid=669875208&adgroupid=36667840951&matchtype=e&device=c&keyword=jeevansathi&gclid=CjwKCAjw-sqKBhBjEiwAVaQ9a3VGwgE648WCwwlF_uBpj49HpXlYP2d40yVVScURnubnNhywJk5ZyxoC8QcQAvD_BwE>

<https://www.naukri.com/registration/createAccount?othersrcp=22636>

HTML Attributes:

<https://www.tutorialspoint.com/html/html_attributes.htm>

<https://www.w3schools.com/html/html_attributes.asp>

# Locators

There are **8 ways** to locate a WebElements

1. lD
2. Name
3. ClassName (Class)
4. Link Text – Anchror Link
5. Partial Link Text
6. CSS Selector
7. Tag Name - driver.findElements(By.tagName(“a”))
8. Xpath

# Xpath

Xpath is used when there is no other locators.

1. Absoute Xpath – start / and from root node /html – if there is a change of element in webpage, which leads the change of absolute xpath
2. Relative Xpath – starts with // - if forms the xpath from the previous nearby element which has id . Even if there is change of element in website most probably which will not make any change in the relative xpath possibility of relative xpath’s working is very high.

**Xpath functions (or) Customized Xpath (or) Dynamic Xpath**

* //\*[@class='mbs \_52lq fsl fwb fcb'] ( in facebook account creation)
* Contains() – look for full or partial attribute value
  + //\*[contains(@name,'first)]
* Text() – exact match with the text which is between the open and closed elements / tags
  + //\*[text()='Request OTP']
* Start-with()
  + //\*[starts-with(@id,’any text here’)]
  + Ex
    - Id = u\_b\_5 (infy.com, name textbox)
    - Id=u\_b\_8
  + Starts-with(@id,’u\_b’)
* Index ( Position() )
  + //\*[@id=’anytext’]/div/a/div[1]/div[position()=3]/p
* And & OR operations

Contains()

Syntax:

//\*[contains(attribute, ‘ ’)]

//\*[contains(text(),’sample text’)]

Website: [www.Freecrm.com](http://www.Freecrm.com)

//a[contains(text(),'Sign Up')]

//a[text()='Sign Up']

//input[starts-with(@placeholder,'E-mail')]

**Xpath Axes**

Axes means relationships of current context node

Purpose:

1. Dynamic Xpath
2. To get all the elements with the Xpath

parent child

ancestor descendant

ancestor-or-self descendant-or-self

preceding following

preceding-sibling following-sibling

//select[@name="birthday\_year"]/parent::div

//select[@name="birthday\_year"]/child::div

//select[@name="birthday\_year"]/ancestor::div

//\*[@id="reg\_form\_box"]/descendant::div

//\*[@id="password\_field"]/ancestor::div[2]

The following is the example for Xpath Axes in **Amazon** website for **laptop** search

//\*[@id="search"]/div[1]/div[1]/div/span[3]/div[2]/div[5]/following::div

//\*[@id="search"]/div[1]/div[1]/div/span[3]/div[2]/div[7]/preceding::div

# CSS Selector

* [1. ID](https://www.browserstack.com/guide/css-selectors-in-selenium#toc2)
  + #From – From is id
* [2. Class](https://www.browserstack.com/guide/css-selectors-in-selenium#toc3)
  + .btnSubmit – btnSubmit is class name
* [3. Attribute](https://www.browserstack.com/guide/css-selectors-in-selenium#toc4)
  + Input[name=’username’]
* [4. Sub-string](https://www.browserstack.com/guide/css-selectors-in-selenium#toc5)
  + Prefix text - ^
  + Suffix Text - $
  + Sub-string - \*
* [5. Inner text](https://www.browserstack.com/guide/css-selectors-in-selenium#toc6)

# Wait (Test / Page Synchronization)

**Types of Wait:**

1. Thread.sleep()
2. Implicit Wait
3. Explicit Wait
4. Fluent Wait

Few other waits:

1. Thread.sleep()
2. Pageloadtimeout()

This sets the time to wait for a page to load completely before throwing an error. If the timeout is negative, page loads can be indefinite.

* The pageLoadTimeout limits the time that the script allots for a web page to be displayed
* If the page loads within the time then the script continues
* If the page does not load within the timeout the script will be stopped by a TimeoutException

**Syntax:**

driver.manage().timeouts().pageLoadTimeout(100, SECONDS);

1. setScriptTimeout()

This is used to set the amount of time the WebDriver must wait for an asynchronous script to finish execution before throwing an error. If the timeout is negative, then the script will be allowed to run indefinitely.

The default timeout for **setScriptTimeout** method is zero. If you do not set time, then there are chances that **executeAsyncScript** method may fail because the JavaScript code may take more than the time allotted. To avoid such failures, set the setScriptTimeout. This is mainly used for Javascript objects and executors.

Implicitly Wait -> DOM Structure -> JavaScript

E.g:

driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS)

1. Explicit Wait -> 60, condition (profile image) – default – every sec

E.g:

WebDriverWait wait = new WebDriverWait(driver,50);

wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*xpath*("/html/body/div[1]/div[2]/div/section[1]/div[1]/div[3]/div[2]/div/span")));

1. Fluent Wait 0 > 60, 5, condition (profile image

Eg:

Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)

.withTimeout(20,TimeUnit.*SECONDS*)

.pollingEvery(4, TimeUnit.*SECONDS*)

.ignoring(NoSuchElementException.class);

<http://www.worldslongestwebsite.com/>

<https://alaskatrips.poweredbygps.com/>

<https://www.softwaretestingmaterial.com>

Implicit Wait:

* Wait until getting DOM Structure is loaded (not necessary to wait for all the web elements to be loaded)
* [Implicit Wait](http://seleniumhq.org/docs/04_webdriver_advanced.html#implicit-waits) - It instructs the web driver to wait for some time by poll the DOM. Once you declared implicit wait it will be available for the entire life of web driver instance. By default the value will be 0. If you set a longer default, then the behavior will poll the DOM on a periodic basis depending on the browser/driver implementation.
* Throws a NoSuchElementException when the element is not present in the DOM
* Throws a ElementNotVisibleException when element is present in the DOM, however, it is hidden and cannot be interacted with

Explicit Wait:

* Explicit wait in Selenium throws a WebDriverTimeoutException, depending on your expected condition

Fluent Wait:

The Fluent Wait command defines the maximum amount of time for Selenium WebDriver to wait for a certain condition to appear. It also defines the frequency with which WebDriver will check if the condition appears before throwing the “ElementNotVisibleException”.

To put it simply, Fluent Wait looks for a web element repeatedly at regular intervals until timeout happens or until the object is found.

Fluent Wait commands are most useful when interacting with web elements that can sometimes take more time than usual to load. This is largely something that occurs in Ajax applications.

While using Fluent Wait, it is possible to set a default polling period as needed. The user can configure the wait to ignore any exceptions during the polling period.

 Wait wait = new FluentWait(driver)

    .withTimeout(30, SECONDS)

    .pollingEvery(5, SECONDS)

    .ignoring(NoSuchElementException.class);

  WebElement foo = wait.until(new Function() {

    public WebElement apply(WebDriver driver) {

    return driver.findElement(By.id("foo"));

    }

   });

**Note:** Implicit, Explicit and Fluent waits,  are dynamic waits. What are dynamic waits? Consider a situation where you have given TimeOut value as 20 seconds. If the element is loaded in 5 seconds, then rest 15 seconds will be ignored. It won’t wait till TimeOut value is completed i.e 20 seconds. That’s why all waits are considered as dynamic waits.

<https://www.swtestacademy.com/selenium-webdriver-wait/>

Solutions: We always get confuse when it comes to using Wait commands, to better understand it we need to remember that there is a difference between several scenarios:

An element not being present at all in the DOM.

An element being present in the DOM but not visible. (Ex. image and Video loading)

An element being present in the DOM but not enabled. (i.e. In the form, submit button will be enabled only after filling mandatory fields)

There are pages which get displayed with the JavaScript, the elements are already present in the browser DOM, but are not visible. The implicit wait only waits for an element to appear in the DOM, so it returns immediately, but when you try to interact with the element you get a NoSuchElementException. You could test this hypothesis by writing a helper method that explicit wait for an element to be visible or clickable.

|  |  |
| --- | --- |
| **Implicit Wait** | **Explicit Wait** |
| * Implicit Wait time is applied to all the elements in the page | * Explicit Wait time is applied only to those elements which are intended by us |
| * In Implicit Wait, we need **not** specify “ExpectedConditions” on the element to be located | * In Explicit Wait, we need to specify “ExpectedConditions” on the element to be located |
| * It is recommended to use when the elements are located with the time frame specified in Selenium implicit wait | * It is recommended to use when the elements are taking long time to load and also for verifying the property of the element like(visibilityOfElementLocated, elementToBeClickable,elementToBeSelected) |

# Visibility of WebElement

1. isDisplayed()
2. isEnabled()
3. isSelected()

# Window Handles

1. getWindowHandle
2. getWindowHandles

# Web Table

How to retrieve the data from table

<https://money.rediff.com/gainers/bsc/daily/groupa>

# Browser Navigation Commands

1. navigate().to(“www.amazon.com”)
2. navigate.back()
3. navigate.forward()
4. navigate.refresh()

# Actions Class (Keyboard and Mouse activities)

Day 2: 17-Nov-2020

Two Classes:

1. Actions Class

**Actions** class is based on builder design pattern which builds a composite actions

1. Action Class

**Action interface** is only used to represent the single user interaction i.e to perform the series of action items build by Actions class.

1. Keyboard Activities
   1. Alt
   2. Ctrl
   3. Shift
   4. Enter
   5. UpArrow, DownArrow, LeftArrow and Right Array
   6. Function Keys
2. Mouse Activities / Events
   1. Click
   2. Right Click
   3. moveToElemenet
   4. Double Click
   5. Drag and Drop
   6. Click and Hold
   7. Move By Offset

Keyboard Actions:

Resources: <https://www.toolsqa.com/selenium-webdriver/keyboard-events-in-selenium/>

Practise: <https://demoqa.com/text-box>

Below is the code inn normal way

driver.get("https://demoqa.com/text-box");

        // Enter the Full Name

        WebElement fullName = driver.findElement(By.id("userName"));

        fullName.sendKeys("Mr.Peter Haynes");

        //Enter the Email

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

        email.sendKeys("PeterHaynes@toolsqa.com");

        // Enter the Current Address

        WebElement currentAddress=driver.findElement(By.id("currentAddress"));

        currentAddress.sendKeys("43 School Lane London EC71 9GO");

        // Copy the Current Address

        currentAddress.sendKeys(Keys.CONTROL);

        currentAddress.sendKeys("A");

        currentAddress.sendKeys(Keys.CONTROL);

        currentAddress.sendKeys("C");

        //Press the TAB Key to Switch Focus to Permanent Address

        currentAddress.sendKeys(Keys.TAB);

        //Paste the Address in the Permanent Address field

        WebElement permanentAddress=driver.findElement(By.id("permanentAddress"));

        permanentAddress.sendKeys(Keys.CONTROL);

        permanentAddress.sendKeys("V");

Methods Available in Selenium Actions Class:

**Keyboard Events Using Selenium Actions Class API:**

The Keyboard interface has the below mentioned methods:

* sendKeys(keysToSend) : sends a series of keystrokes onto the element
* keyDown(theKey) : Sends a key press without release it. Subsequent actions may assume it as pressed. (example: Keys.ALT, Keys.SHIFT, or Keys.CONTROL)
* keyUp(theKey): Performs a key release

Below is the code which uses Keyboard activities:

driver.get("https://demoqa.com/text-box");

        //Create object of the Actions class

        Actions actions = new Actions(driver);

        // Enter the Full Name

        WebElement fullName = driver.findElement(By.id("userName"));

        fullName.sendKeys("Mr.Peter Haynes");

        //Enter the Email

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

        email.sendKeys("PeterHaynes@toolsqa.com");

        // Enter the Current Address

        WebElement currentAddress=driver.findElement(By.id("currentAddress"));

        currentAddress.sendKeys("43 School Lane London EC71 9GO");

        // Select the Current Address using CTRL + A

        actions.keyDown(Keys.CONTROL);

        actions.sendKeys("a");

        actions.keyUp(Keys.CONTROL);

        actions.build().perform();

        // Copy the Current Address using CTRL + C

        actions.keyDown(Keys.CONTROL);

        actions.sendKeys("c");

        actions.keyUp(Keys.CONTROL);

        actions.build().perform();

        //Press the TAB Key to Switch Focus to Permanent Address

        actions.sendKeys(Keys.TAB);

        actions.build().perform();

        //Paste the Address in the Permanent Address field using CTRL + V

        actions.keyDown(Keys.CONTROL);

        actions.sendKeys("v");

        actions.keyUp(Keys.CONTROL);

        actions.build().perform();

Mouse Movements

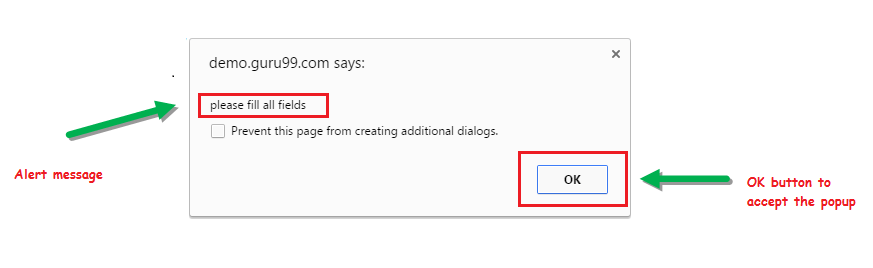
Mouse Events Using Selenium Actions Class API:

* click(): Simply click on element
* doubleClick (): Double clicks onElement
* contextClick() : Performs a context-click (right click) on an element
* clickAndHold(): Clicks at the present mouse location (without releasing)
* dragAndDrop(source, target): Invokes click-and-hold at the source location and moves to the location of the target element before releasing the mouse. source – element to grab, target – element to release
* dragAndDropBy(source, xOffset, yOffset) : Performs click-and-hold at the source location, shifts by a given offset, then frees the mouse. xOffset – to shift horizontally, yOffset – to shift vertically
* moveByOffset(x-offset, y-offset): Shifts the mouse from its current position (or 0,0) by the given offset. x-offset – Sets the horizontal offset (negative value – shifting the mouse to the left), y-offset – Sets the vertical offset (negative value – shifting the mouse to the up)
* moveToElement(toElement): It shifts the mouse to the center of the element
* release(): Releases the depressed left mouse button at the existing mouse location

# Alerts Windows

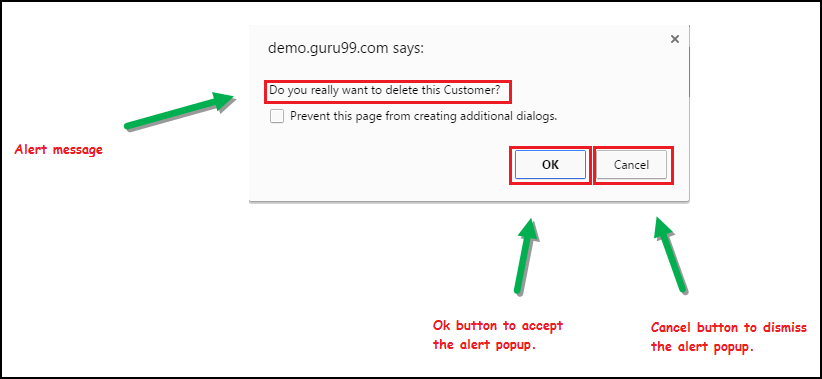
1. Simple Alerts - OK button
2. Confirmation Alert - Ok and Cancel button
3. Prompt Alerts - Text box which will accept some texts.
4. **Simple Alert**

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

[](https://cdn.guru99.com/images/3-2016/032216_1314_AlertPopuph1.png)

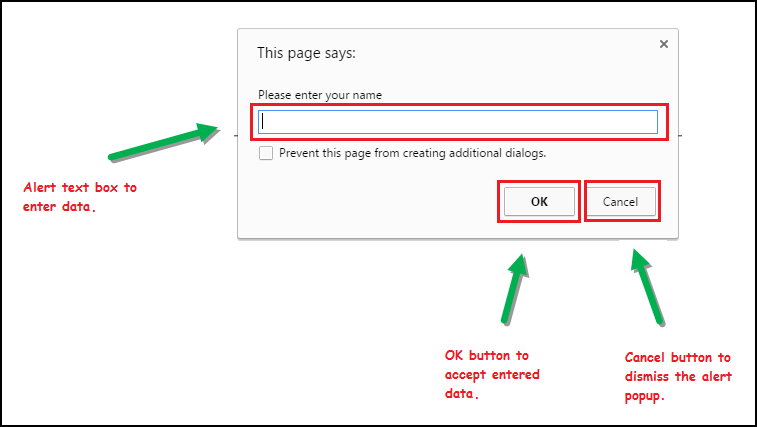
**2) Confirmation Alert.**

**This confirmation alert asks permission to do some type of operation.**

**[](https://cdn.guru99.com/images/3-2016/032216_1314_AlertPopuph3.png)**

**3) Prompt Alert.**

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

[](https://cdn.guru99.com/images/3-2016/032216_1314_AlertPopuph2.png)

**3)**

# JavaScriptExecuter

**What is JavaScriptExecutor?**

JavaScriptExecutor is an Interface that helps to execute[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)through Selenium Webdriver. JavaScriptExecutor provides two methods "executescript" & "executeAsyncScript" to run javascript on the selected window or current page.

**Why do we need JavaScriptExecutor?**

In Selenium Webdriver, locators like XPath, CSS, etc. are used to identify and perform operations on a web page.

In case, these locators do not work you can use JavaScriptExecutor. You can use JavaScriptExecutor to perform an desired operation on a web element.

Two Methods:

1. executeAsyncScript – Asynchronous calls
2. executeScript

Syntax:

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript(Script,Arguments);

executeAsyncScript:

1. Performing a sleep in the browser under test.

executeScript:

1. Generate Alert Window
2. Capture Scrape Data and Navigate to different pages
3. Scroll Down the webpage

<https://www.swtestacademy.com/selenium-javascriptexecutor/>

# Iframes

Iframe is nothing but Inline Frame (webpage) which can be embedded into another webpage. In general ads displayed in the websites are the examples of Iframe.

AJAX – Asychronous Javascript And XML

Why?

The Iframes normally can not be located or identified like locating other Web Elements.

How to switch over the elements in iframes using Web Driver commands:

Basically, we can switch over the elements in frames using 3 ways.

By Index

By Name or Id

By Web Element

# Model Window

<http://uitestpractice.com/>

When a **modal** window opens, it opens inside the current page. ... With **popup** windows, users can often miss them because they don't grab the user's attention. When users click the browser window, the browser window comes to the front and the **popup** window hides behind it.

# Turn off notifications

ChromeOptions ops = new ChromeOptions();

ops.addArguments("--disable-notifications");

driver = new ChromeDriver(ops);

# Take Screen Shot

//Convert web driver object to TakeScreenshot

TakesScreenshot scrShot =((TakesScreenshot)webdriver);

//Call getScreenshotAs method to create image file

File SrcFile=scrShot.getScreenshotAs(OutputType.FILE);

//Move image file to new destination

File DestFile=new File(fileWithPath);

//Copy file at destination

FileUtils.copyFile(SrcFile, DestFile);

# Advanced Selenium

# AutoIT

What is AutoIT?

AutoIt is a freeware scripting language designed for automating windows GUI. It uses a combination of mouse movement, keystrokes and window control manipulation to automate a task which is not possible by selenium webdriver.

Why use AutoIT?

[Selenium](https://www.guru99.com/introduction-to-selenium.html) is an open source tool that is designed to automate web-based applications on different browsers but to handle window GUI and non HTML popups in ap

To handle below elements

Windows authentication box

File uploading dialog box

Any other non-browser interaction

# Robot Class

**What is a Robot class?**

Time plays a major role while testing an application and we need to make sure we complete the desired task within the specific time.

A *Robot class* in [***Selenium***](https://www.edureka.co/blog/what-is-selenium/) is used to generate native system input events for test automation, self-running demos, and other applications where you need control over the mouse and keyboard. ***[WebDriver](https://www.edureka.co/blog/selenium-webdriver-architecture/" \t "_blank)***cannot handle the OS popups, so in Java 1.3, Robot class was introduced.

The primary purpose of this Robot class is to facilitate [***automated testing***](https://www.edureka.co/blog/selenium-tutorial) for Java platform implementations. In simple terms, I would say this class provides control over the mouse and keyboard devices.

This is very easy to implement as it can be easily integrated with current automation framework

Now, let us understand the importance of this class

**Importance of Robot class**

* Uploading a file is easy when we use the robot class
* It can simulate and handle the mouse and keyboard functions
* It can handle pop-ups as well