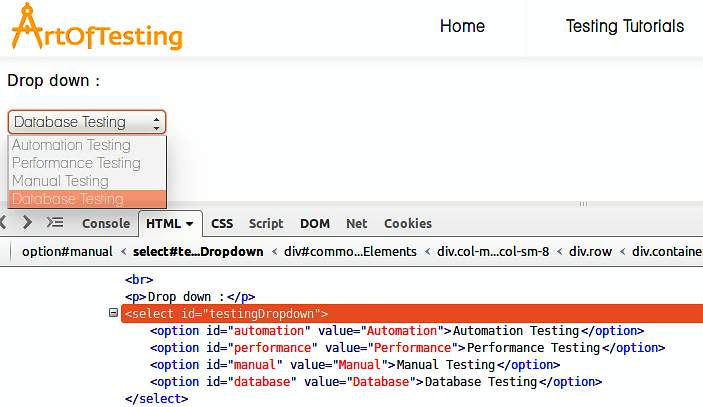
**Handling drop-downs in Selenium WebDriver**

In this tutorial, we are going to study the handling of dropdowns in Selenium WebDriver. For practicing, you can check the [dummy page](http://artoftesting.com/sampleSiteForSelenium.html) having a dropdown element.



## Select in Selenium WebDriver

The 'Select' class in Selenium WebDriver is used for selecting and deselecting option in a dropdown. The objects of Select type can be initialized by passing the dropdown webElement as parameter to its constructor.

WebElement testDropDown = driver.findElement(By.id("testingDropdown"));

Select dropdown = **new** Select(testDropDown);

### Selecting options from dropdown

There are three ways of selecting options from dropdown-

1. **selectByIndex** - To select an option based on its index, beginning with 0.

dropdown.selectByIndex(3);

1. **selectByValue** - To select an option based on its 'value' attribute.

dropdown.selectByValue("Database");

1. **selectByVisibleText** - To select an option based on the text over the option.

dropdown.selectByVisibleText("Database Testing");

### Different utility methods in the Select class

* **DeselectAll()** - To deselect all the selected options.
* **deselectByIndex(int index)** - To deselect the option based on its index.
* **deselectByValue(String valueAttribute)** - To deselect the option its 'value' attribute.
* **deselectByVisibleText(String text)** - To deselect the option based on the text over the option.

**Right Click in Selenium WebDriver**

Hello friends, quite often during automation we need to right click or context click an element. Later, this action is followed up by pressing the UP/DOWN arrow keys and ENTER key to select the desired context menu element (check our tutorial on pressing the non-text keys in selenium - [Pressing ARROW KEYS, FUNCTION KEYS and other non-text keys in Selenium](http://artoftesting.com/sampleSiteForSelenium.html) ).  
For right clicking an element in Selenium, we make use of the Actions class. The Actions class provided by Selenium Webdriver is used to generate complex user gestures including right click, double click, drag and drop etc.

### Code snippet to right click an element

Actions action = **new** Actions(driver);

WebElement element = driver.findElement(By.id("elementId"));

action.contextClick(element).perform();

Here, we are instantiating an object of Actions class. After that, we pass the WebElement to be right clicked as parameter to the contextClick() method present in the Actions class. Then, we call the perform() method to perform the generated action.

# Double Click in Selenium WebDriver

Hello friends! in this post, we will learn to double click an element using Selenium Webdriver with Java. For double clicking an element in Selenium we make use of the Actions class. The Actions class provided by Selenium Webdriver is used to generate complex user gestures including right click, double click, drag and drop etc.

### Code snippet to double click an element

Actions action = **new** Actions(driver);

WebElement element = driver.findElement(By.id("elementId"));

action.doubleClick(element).perform();

Here, we are instantiating an object of Actions class. After that, we pass the WebElement to be double clicked as parameter to the doubleClick() method present in the Actions class. Then, we call the perform() method to perform the generated action.

# Mouseover in Selenium WebDriver

Hello friends! in this post, we will learn to automate the mouseover over an element using Selenium Webdriver with Java. For performing the mouse hover over an element in Selenium, we make use of the Actions class. The Actions class provided by Selenium Webdriver is used to generate complex user gestures including mouseover, right click, double click, drag and drop etc.

### Code snippet to mouseover

Actions action = **new** Actions(driver);

WebElement element = driver.findElement(By.id("elementId"));

action.moveToElement(element).perform();

Here, we are instantiating an object of Actions class. After that, we pass the WebElement to be mouse hovered as parameter to the moveToElement() method present in the Actions class. Then, we will call the perform() method to perform the generated action.

# Scrolling in Selenium Webdriver

Scrolling a webpage is required in automation when the application requires scrolling down or up to display additional information e.g. most of the e-commerce sites display only 10-20 products at a time and then load more products as the user scrolls down. In this tutorial, we'll take example of an e-commerce website - flipkart and automate the scrolling of webpage in order to fetch more results.  
  
In automation first we will launch [filpkart.com](http://filpkart.com/) , write a search term and then scroll down to fetch more results corresponding to that search term. Automating page scrolling will make use of "scrollBy" method of javascript. For executing the javascript method we will use Javascript executor. The scrollBy method takes two parameters one each for horizontal and vertical scroll in terms of pixels.

JavascriptExecutor js = (JavascriptExecutor)driver;

js.executeScript("scrollBy(0, 2500)");

Following test script automates flipkart's scroll down functionality to test display of new pages on the search result page as the user scrolls down.

**Navigate back and forward in browser**

During automation we are at times required to move back to the previuos page or move forward to the next page in browser history. In this post we will learn to perform these operations using driver.navigate command.

## Navigate back in browser history

driver.navigate().back();

## Navigate forward in browser history

driver.navigate().forward();

# Refresh a webpage in selenium

There are multiple ways of refreshing a page in Selenium Webdriver. In this post, we will present all these ways and also specify which one is the best.

1. Using driver.navigate command-  
   Selenium Webdriver provides inherent support for refreshing a webpage using its driver.navigate command. This is by far the most preferred and widely used way of refreshing a webpage.

driver.navigate().refresh();

1. Opening current URL using driver.getCurrentUrl() with driver.get() command-

driver.get(driver.getCurrentUrl());

1. Opening current URL using driver.getCurrentUrl() with driver.navigate() command-

driver.navigate().to(driver.getCurrentUrl());

1. Pressing F5 key on any textbox using sendKeys command-

driver.findElement(By textboxLocator).sendKeys(Keys.F5);

1. Passing ascii value of F5 key i.e. "\uE035" using sendKeys command-

driver.findElement(By textboxLocator).sendKeys("\uE035");

**Maximize or Minimize a browser**

When we launch a browser using Selenium WebDriver, by default it is not in its maximized state. In this post, we will see how to maxmize and minimize a browser during automation.

## Maxmize a browser

During automation its one of the best practices to maximize the browser in the initial phases (or in the @BeforeTest method in case you are using TestNG as the testing framework). The following command can be used to maximize the browser window.

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

## Minimize a browser

Unfortunately Selenium WebDriver doesn't provide any native supoort for minimizing a browser. If we don't want to see the browser in action during automation than we can **use Headless browsers like HTMLUnitDriver, PhantomJS** etc. But in case we specifically want to minimize a browser during automation than we can make use of the below statement. The following code snippet will position the browser to an area that is not within the viewable section of the machine we are working on. Thus making it appear as if the browser window got minimized.

driver.manage().window().setPosition(**new** Point(**0**, -**1000**));