**Top 25 Selenium WebDriver Commands That You Should Know**

Last Updated:[March 11, 2020](https://www.softwaretestinghelp.com/selenium-webdriver-commands-selenium-tutorial-17/)

**Top Selenium WebDriver Commands – A Phenomenal Guide for Automation Testers**

Selenium WebDriver is one of the most popular open source website automation tools. Most of my fellow automation testers prefer the combination of WebDriver with Java.

***In this tutorial, I will discuss 25 routinely used Selenium WebDriver commands along with their concerned Syntax and simple examples for your easy understanding.***



**What You Will Learn:**[[show](https://www.softwaretestinghelp.com/selenium-webdriver-commands-selenium-tutorial-17/)]

Types Of Commands in WebDriver

In the **last Selenium tutorial**, we discussed the [different types of alerts](https://www.softwaretestinghelp.com/handle-alerts-popups-selenium-webdriver-selenium-tutorial-16/)

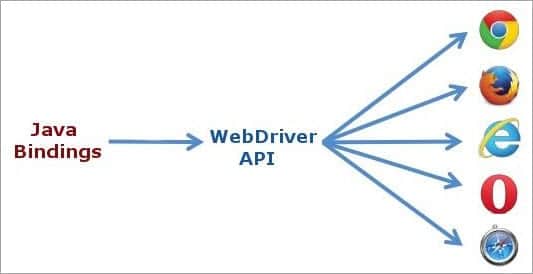
encountered while testing web-based applications and their effective ways of handling. We discussed both the types of alerts i.e. “Web-based alerts” and “Window-based alerts” at length. We also made you acquainted with yet another Java-based utility named as “Robot Class” to handle Windows-based pop-up.

Advancing ahead in [this Selenium WebDriver tutorial series](https://www.softwaretestinghelp.com/selenium-tutorial-1/), we would be pressing on **various commonly and routinely used Selenium WebDriver commands**. We will precisely and briefly discuss each of these Selenium commands so as to make you capable of using these commands effectively whenever the situation arises.

Every automation Java work file starts with creating a reference of web browser that we wish to use as mentioned in the below syntax.

[Webdriver command](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2018/04/Webdriver-command.jpg)

There are several methods that are available from the Webdriver interface. These methods are accessed using the instance variable **driver** in a simple format **driver.methodName();**. All these automation projects include calling these methods and comparing & evaluating what they actually return.

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2018/04/Webdriver-api.jpg)

**In simple terms, we can generally classify Webdriver commands as:**

* Browser commands,
* Get commands,
* Navigation commands,
* Webelement commands,
* Action commands and
* Result commands.

From the context of manual testing, the result of a test, either PASS or FAIL is decided from the Result commands which usually compares the expected & actual results and the rest are Testcase steps.

Top 7 Selenium Commands with Details

**Just to have a rough idea, we would be discussing the following Selenium WebDriver commands and their different versions:**

1. **get()** methods
2. Locating links by **linkText()**and **partialLinkText()**
3. Selecting multiple items in a drop dropdown
4. Submitting a form
5. Handling iframes
6. **close()**and **quit()** methods
7. Exception Handling

**#1) get() Methods**

| **WebDriver command** | **Usage** |
| --- | --- |
| get() | • The command launches a new browser and opens the specified URL in the browser instance • The command takes a single string type parameter that is usually a URL of application under test • To the Selenium IDE users, the command may look very much like open command  driver.get("https://google.com"); |
| getClass() | The command is used to retrieve the Class object that represents the runtime class of this object  driver.getClass(); |
| getCurrentUrl() | • The command is used to retrieve the URL of the webpage the user is currently accessing • The command doesn’t require any parameter and returns a string value  driver.getCurrentUrl(); |
| getPageSource() | • The command is used to retrieve the page source of the webpage the user is currently accessing • The command doesn’t require any parameter and returns a string value • The command can be used with various string operations like contains() to ascertain the presence of the specified string value  boolean result = driver.getPageSource().contains("String to find"); |
| getTitle() | • The command is used to retrieve the title of the webpage the user is currently working on. A null string is returned if the webpage has no title • The command doesn’t require any parameter and returns a trimmed string value  String title = driver.getTitle(); |
| getText() | • The command is used to retrieve the inner text of the specified web element • The command doesn’t require any parameter and returns a string value • It is also one of the extensively used commands for verification of messages, labels, errors etc displayed on the web pages.  String Text = driver.findElement(By.id("Text")).getText(); |
| getAttribute() | • The command is used to retrieve the value of the specified attribute • The command requires a single string parameter that refers to an attribute whose value we aspire to know and returns a string value as a result.  driver.findElement(By.id("findID")). getAttribute("value"); |
| getWindowHandle() | • The command is used to tackle with the situation when we have more than one window to deal with. • The command helps us switch to the newly opened window and performs actions on the new window. The user can also switch back to the previous window if he/she desires.  private String winHandleBefore; winHandleBefore = driver.getWindowHandle(); driver.switchTo().window(winHandleBefore); |
| getWindowHandles() | • The command is similar to that of “getWindowHandle()” with the subtle difference that it helps to deal with multiple windows i.e. when we have to deal with more than 2 windows. |

**The code snippet for “getWindowHandles()” is given below:**

public void explicitWaitForWinHandle(final WebDriver dvr, int timeOut, final boolean close) throws WeblivException

{

try {

Wait<WebDriver> wait = new WebDriverWait(dvr, timeOut);

ExpectedCondition<Boolean> condition = new ExpectedCondition<Boolean>() {

@Override

public Boolean apply(WebDriver d) {

int winHandleNum = d.getWindowHandles().size();

if (winHandleNum > 1)

{

// Switch to new window opened

for (String winHandle : d.getWindowHandles())

{

dvr.switchTo().window(winHandle);

// Close the delete window as it is not needed

if (close && dvr.getTitle().equals("Demo Delete Window"))

{

dvr.findElement(By.name("ok")).click();

}

}

return true;

}

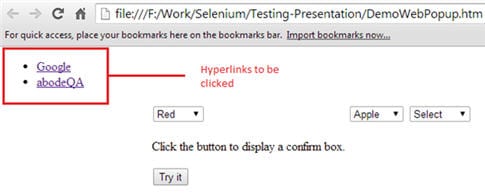
return false;

}

};

#2) Locating links by linkText() and partialLinkText()

Let us access “google.com” and “abodeqa.com” using *linkText()* and *partialLinText()* methods of WebDriver.

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/Selenium-commands-1.jpg)

**The above mentioned links can be accessed by using the following commands:**

*driver.findElement(By.linkText(“Google”)).click();*  
*driver.findElement(By.linkText(“abodeQA”)).click();*

The command finds the element using link text and then click on that element and thus the user would be re-directed to the corresponding page.

**The above-mentioned links can also be accessed by using the following commands:**

*driver.findElement(By.partialLinkText(“Goo”)).click();*  
*driver.findElement(By.partialLinkText(“abode”)).click();*

The above two commands find the elements based on the substring of the link provided in the parenthesis and thus partialLinkText() finds the web element with the specified substring and then clicks on it.

#3) Selecting multiple items in a drop dropdown

There are primarily two kinds of dropdowns:

1. **Single select dropdown**: A drop-down that allows only single value to be selected at a time.
2. **Multi-select dropdown**: A drop-down that allows multiple values to be selected at a time.

**Consider the HTML code below** for a dropdown that can select multiple values at the same time.

<select id="SelectID\_One" multiple="">

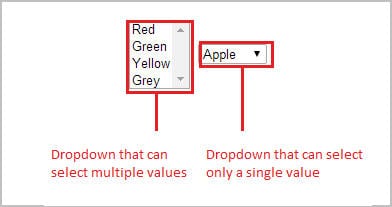
<option value="redvalue">Red</option>

<option value="greenvalue">Green</option>

<option value="yellowvalue">Yellow</option>

<option value="greyvalue">Grey</option>

</select>

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/Selenium-commands-2.jpg)  
The code snippet below illustrates the multiple selections in a drop down.

// select the multiple values from a dropdown

Select selectByValue = new Select(driver.findElement(By.id("SelectID\_One")));

selectByValue.selectByValue("greenvalue");

selectByValue.selectByVisibleText("Red");

selectByValue.selectByIndex(2);

#4) Submitting a form

Most or almost all the websites have forms that need to be filled and submitted while testing a web application. The user may come across several types of forms like Login form, Registration form, File Upload form, Profile Creation form etc.

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/Selenium-commands-3.jpg)

In WebDriver, a user is leveraged with a method that is specifically created to submit a form. The user can also use click method to click on the submit button as a substitute to submit button.

**Check out the code snippet below against the above “new user” form:**

// enter a valid username

driver.findElement(By.<em>id</em>("username")).sendKeys("name");

// enter a valid email address

driver.findElement(By.<em>id</em>("email")).sendKeys("name@abc.com");

// enter a valid password

driver.findElement(By.<em>id</em>("password")).sendKeys("namepass");

// re-enter the password

driver.findElement(By.<em>id</em>("passwordConf")).sendKeys("namepass");

// submit the form

driver.findElement(By.<em>id</em>("submit")).submit();

Thus, as soon as the program control finds the submit method, it locates the element and triggers the *submit()* method on the found web element.

#5) Handling iframes

While automating web applications, there may be situations where we are required to deal with multiple frames in a window. Thus, the test script developer is required to switch back and forth between various frames or iframes for that matter of fact.

An inline frame acronym as iframe is used to insert another document within the current HTML document or simply a web page into another web page by enabling nesting.

**Consider the following HTML code having the iframe within the webpage:**

<html>

<head><title>Software Testing Help - iframe session</title>

</head>

<body>

<div>

<iframe id="ParentFrame">

<iframe id="ChildFrame">

<input type="text" id="Username">UserID</input>

<input type="text" id="Password">Password</input>

</iframe>

<button id="LogIn">Log In</button>

</iframe>

</div>

</body>

</html>

The above HTML code illustrates the presence of an embedded iframe into another iframe. Thus, to be able to access the child iframe, a user is required to navigate to the parent iframe first. After performing the required operation, a user may be required to navigate back to the parent iframe to deal with the other element of the webpage.

It is impossible if a user tries to access the child iframe directly without traversing to the parent iframe first.

**Select iframe by id**  
*driver.switchTo().frame(“ID of the frame“);*

**Locating iframe using tagName**

While locating an iframe, the user might face some trouble if the iframe is not attributed with standard properties. It becomes a complex process to locate the frame and switch to it. To buckle down the situation, a user is leveraged to locate an iframe using a tagName method similar to the way we find any other web element in WebDriver.

*driver.switchTo().frame(driver.findElements(By.tagName(“iframe”).get(0));*

The above command locates the first web element with the specified tagName and switches over to that iframe. “get(0) is used to locate the iframe with the index value.” Thus, in lines with our HTML code, the above code syntax would lead the program control to switch to “ParentFrame”.

**Locating iframe using the index:**

**a) frame(index)**  
*driver.switchTo().frame(0);*

**b) frame(Name of Frame)**  
*driver.switchTo().frame(“name of the frame”);*

**c) frame(WebElement element)**  
Select Parent Window  
*driver.switchTo().defaultContent();*

The above command brings the user back to the original window i.e. out of both the iframes.

#6) close() and quit() methods

There are two types of commands in WebDriver to close the web browser instance.

**a) close()**: WebDriver’s close() method closes the web browser window that the user is currently working on or we can also say the window that is being currently accessed by the WebDriver. The command neither requires any parameter nor does it return any value.

**b) quit()**: Unlike close() method, quit() method closes down all the windows that the program has opened. Same as close() method, the command neither requires any parameter nor does it return any value.

**Refer the below code snippets:**

*driver.close();* // closes only a single window that is being accessed by the WebDriver instance currently  
*driver.quit();* // closes all the windows that were opened by the WebDriver instance

#7) Exception Handling

Exceptions are the conditions or situations that halt the program execution unexpectedly.

**Reasons for such conditions can be:**

* Errors introduced by the user
* Errors generated by the programmer
* Errors generated by physical resources

Thus, to deal with these unexpected conditions, exception handling was conceptualized.

With respect to Java code that we implement while automating a web application can be enclosed within a block that that is capable of providing a handling mechanism against the erroneous conditions.

**Catching an exception**

To catch an exception, we use the below block of code

try{

// Protected block

// implement java code for automation

}

catch (ExceptionName e)

{

// catch block - Catches the exceptions generated in try block without halting the program execution

}

If any exception occurs in the try block/protected block, then the execution controls checks for a catch block for the matching exception type and passes the exception to it without breaking the program execution.

**Multiple Catch Blocks**

try{

//Protected block

}

catch (ExceptionType1 e)

{

// catch block

}

catch (ExceptionType2 e)

{

// catch block

}

catch (ExceptionType3 e)

{

// catch block

}

In the above code, an exception is likely to be caught in the first catch block if the exception type matches. If the exception type does not match, then the exception is traversed to the second catch block and third catch block and so on until the all catch blocks are visited.

**WebDriver conditions and Exception Handling**

When we aspire to verify the presence of any element on the webpage using various WebDriver ‘s conditional commands, WebDriver presumes the web element to be present on the web page. If the web element is not present on the web page, the conditional commands throw a “NoSuchElementPresentException”. Thus to avoid such exceptions from halting the program execution, we use Exception Handling mechanisms. Refer the code snippet below:

WebElement saveButton = driver.findElement(By.id("Save"));

try{

if(saveButton.isDisplayed()){

saveButton.click();

   }

}

catch(NoSuchElementException e){

e.printStackTrace();

}

List of 25 More Popular WebDriver Commands & Examples

Given below is the list of top 25 routinely used Webdriver Commands that every Automation Tester must know.

***Let’s go through these commands one by one in detail.***

**Further reading =>**[**Selenium WebDriver Tutorial**](https://www.softwaretestinghelp.com/selenium-webdriver-selenium-tutorial-8/)

#1) get()

**Command using get() to open a URL in the current browser.**

The command below will open the specified URL, ‘https://www.softwaretestinghelp.com' in the browser.

**Syntax:**

driver.get("https://www.softwaretestinghelp.com");

**Explanation:**

* Navigate to the URL https://www.softwaretestinghelp.com

#2) getCurrentUrl()

**Command using getCurrentUrl() to check if the URL is correct.**

The below command gets the current URL in the string format.

**Syntax:**

driver.getCurrentUrl();

We usually use this method in commands to check if we have navigated to the right page as expected. For that, we have to use Assert as shown in the below **Example**.

**Syntax:**

Assert.assertEquals(expectedUrl,  driver.getCurrentUrl());

Where expectedUrl is the URL that is expected in the string format.

**Explanation:**

* Check and verify that the URL loaded remains the same and the correct page is loaded.

#3) findElement(By, by) and click()

**findElement(By, by) and click() to Click on any element of the webpage.**

The findElement(By, by) method searches and locates the first element on the current page, which matches the criteria given as a parameter. This method is usually used in commands to simulate user actions like click, submit, type etc.

The below command searches and locates the first element in the webpage with id”submit1” and clicks on it if it is not covered.

**Syntax:**

driver.findElement(By.id("submit1")).click();

The element can be located using **ID**, **Name**, **Class** **Name**, **Tag Name**, **Link Text &** **Partial Link Text**, **CSS Selector** and **X Path**.

**Explanation:**

* Look for the required Submit button.
* Click on the button.

The command below selects an item from the list box.

**Syntax:**

WebElement roleDropdown = driver.findElement(By.id("name1");

roleDropdown.click();

**Explanation:**

* Search and locate the list item by id “name1”.
* Click on that item.

#4) isEnabled()

**isEnabled() to Check Whether the Element is Enabled Or Disabled in the Selenium WebDriver.**

In order to check if a particular element is enabled in a web page, we use isEnabled() method.

**Syntax:**

boolean textBox = driver.findElement(By.xpath("//input[@name='textbox1']")).isEnabled();

**Explanation:**

* Finds the element in the webpage according to the xpath and checks if the element is enabled.

#5) findElement(By, by) with sendKeys()

**findElement(By, by) with sendKeys() to type in the form fields.**

Form validation checks by entering the different user inputs that are often required in automation testing. We use findElement(By, by) to locate the fields and sendKeys() to type some content into an editable field.

The below command uses the Name locator to find the form field and types “Aaron” in it.

**Syntax:**

driver.findElement(By.name("name")).sendkeys("Aaron");

**Explanation:**

* Look for the required name field in the form.
* Enter the value “Aaron” in it.

#6) findElement(By, by) with getText()

**findElement(By, by) with getText() to store value of targeted web element.**

The getText() is a method that gets you the inner text of the web element. Get text is the text inside the HTML tags.

The below code finds the Element with tagName “select” and gets the text inside the tag and stores it in a variable drop-down. Now the String dropDown can be used for further actions inside the program.

**Syntax:**

String dropDown = driver.findElement(By.tagName("dropdown1")).getText();

**Explanation:**

* Look for the required field in the form which has the tagName “dropdown1”.
* Take the text inside its HTML tag.
* Store the text in String object ‘DropDown’.

#7) Submit()

**Submit() to submit a web form.**

The click() method that we discussed above can be used to click on any links or buttons. Submit() is a better alternative to click() if the element to be clicked is a submit button. The submit button is inside the HTML ‘form’ tag and the type of button is ‘submit’(not ‘button’).

The submit() makes life easier by automatically finding the button and the method that can be appended to any other field like name or email address. In the case of click, we have to use findElement(By, by) method and specify the correct locators.

In some scenarios where the action is done through elements other than a button, submit() works and click() won't.

**Syntax:**

driver.findElement(By.xpath("//input[@name='comments']")).submit();

**Explanation:**

* Find element in the given x path with name ‘comments’.
* Submit the form.

#8) findElements(By, by)

**findElements(By, by) to get the list of web elements.**

Sometimes we might want to print or do an action on a list of web elements like links or input fields in a webpage. In such a case, we use findElements(By, by).

**Syntax:**

List<WebElement> allChoices = dropDown.findElements(By.xpath(".//fruitoption"));

**Explanation:**

* A list of all the web elements with specified xpath is stored in the webelement list allChoices.

#9) findElements(By, by) with size()

**findElements(By, by) with size() to verify if an element is present.**

findElements(By, by) can be used to verify if an element is actually present in the webpage.

The command below is used if we want to verify that an element with particular locator is present in a webpage. If size() != 0 then the element is present.

**Syntax:**

Boolean checkIfElementPresent= driver.findElements(By.xpath("//input[@id='checkbox2']")).size()!= 0;

**Explanation:**

* Find element is specified in xpath with id ‘checkbox2’.
* According to the size of the element list, the Boolean checkIfElementPresent will be set to TRUE or FALSE.

#10) pageLoadTimeout(time,unit)

**pageLoadTimeout(time,unit) to set the time for a page to load.**

Sometimes due to server issues or network delays, a page might take more than usual time to load. This might throw an error in the program. In order to avoid this, we set a wait time and pageLoadTimeout() is one of such method. This will usually follow a get() command.

**Syntax:**

driver.manage().timeouts().pageLoadTimeout(500, SECONDS);

**Explanation:**

* Wait for 500 seconds for a page to load.

#11) implicitlyWait()

**implicitlyWait() to set a wait time before searching and locating a web element.**

What happens if the Webdriver tries to locate an element before the webpage loads and the element appears? NoSuchElementExeption will be thrown. In order to avoid this, we can add a command to implicitly wait for a certain amount of time before locating the element.

**Syntax:**

driver.manage().timeouts().implicitlyWait(1000, TimeUnit.SECONDS);

**Explanation:**

* Implicitly wait for 1000 seconds before executing the next line in the code.

#12) untill() and visibilityOfElementLocated()

**untill() from WebdriverWait and visibilityOfElementLocated() from ExpectedConditions to wait explicitly till an element is visible in the webpage.**

To handle cases where an element takes too much time to be visible on the software web page applying implicit wait becomes tricky. In this case, we can write a comment to wait until the element appears on the webpage. This command uses a combination of until() method from the WebdriverWait Class and visibilityOfElementLocated() method from the ExpectedConditions class.

**Syntax:**

WebDriverWait wait = new WebDriverWait(driver, 10);

WebElement element = wait.until(ExpectedConditions.visibilityOfElementLocated (By.xpath("//input[@id=’name’]")));

**Explanation:**

* The first line says how much time to wait which is 10 seconds.
* The second condition says an expected condition to wait for. Here it is an element with id’name’ in the mentioned xpath.

#13) untill() and alertIsPresent()

**untill() from WebdriverWait and alertIsPresent() from ExpectedConditions to wait explicitly till an alert appears.**

In some scenarios, we have to wait for alerts to continue the test. In this case, we use a command using until() method from the WebdriverWait class and alertIsPresent() method from the ExpectedConditions class.

**Please see the command below:**

WebDriverWait wait = new WebDriverWait(driver, 10);

WebElement element = wait.until(ExpectedConditions.alertIsPresent()

);

**Explanation:**

* The first line says how much time to wait – that is 10 seconds.
* The second condition says an expected condition to wait for. Here it is an alert pop up.

#14) getTitle()

**getTitle() to get page title in the Selenium webdriver.**

**Syntax:**

String title = driver.getTitle();

System.out.println(title);

This is usually used to print the title in the output logs.

**Explanation:**

* Get the title of the webpage and store it in the String object title.
  + Print the value stored in the title to the output logs.

#15) Select

**Select class for selecting and deselecting values from the drop-down in Selenium WebDriver.**

We often have dropdown related scenarios. Methods from the Select class is used to handle this. We can use selectByVisibleText(),selectByValue() or selectByIndex() according to the scenario.

**Syntax:**

WebElement mySelectedElement = driver.findElement(By.id("select"));

Select dropdown= new Select(mySelectedElement);

dropdown.selectByVisibleText("Apple");

**Explanation:**

* Find Drop down using it’s id “select”.
* Select the visible text “Apple” from the dropdown.

WebElement mySelectedElement = driver.findElement(By.id("select"));

Select dropdown= new Select(mySelectedElement);

Dropdown.selectByValue("Apple")

**Explanation:**

* Find the Drop down using it’s id “select”.
* Select the text with value “Apple” from the dropdown.

WebElement mySelectedElement = driver.findElement(By.id("select"));

Select dropdown= new Select(mySelectedElement);

listbox.selectByIndex(1);

**Explanation:**

* Find the Drop down using it’s id “select”.
* Select the drop-down item with index value ‘1’ from the drop-down (Second item).

Similar to the select, we can deselect values from the drop-down using similar commands.

**Please check the commands:**

WebElement mySelectedElement = driver.findElement(By.id("select"));

Select dropdown= new Select(mySelectedElement);

dropdown.deselectByVisibleText("Apple");

**Explanation:**

* Find the Drop down using it’s id “select”.
* Deselect the visible text “Apple” from the drop-down.

WebElement mySelectedElement = driver.findElement(By.id("select"));

Select dropdown= new Select(mySelectedElement);

Dropdown.deselectByValue("Apple");

**Explanation:**

* Find the Drop down using it’s id “select”.
* De-select the text with value “Apple” from the drop-down.

WebElement mySelectedElement = driver.findElement(By.id("select"));

Select dropdown= new Select(mySelectedElement);

listbox.deselectByIndex(1);

**Explanation:**

* Find the Drop down using it’s id “select”.
* De-select the drop-down item with the index value ‘1’ from the drop-down (Second item).

#16) navigate()

**navigate() to navigate between the URLs.**

We often see scenarios were we might want to navigate from the landing URL and then go back or forward. In such cases, instead of using get(), we can use navigate(). In Navigate we can use back() and forward() methods without specifying the URLs.

**Syntax:**

driver.navigate().to("https://www.softwaretestinghelp.com");

driver.navigate().back();

driver.navigate().forward();

**Explanation:**

* Navigate to https://www.softwaretestinghelp.com
* Navigate back.
* Navigate forward.

#17)  getScreenshotAs()

**getScreenshotAs() to Capture the entire page screenshot in Selenium WebDriver.**

This one is often required to save your work details or sometimes to manually check the outputs. The below command is used to take a screenshot and save in an output file.

**Syntax:**

File shot = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(shot, new File("D:\\ shot1.jpg"));

**Explanation:**

* Take a screenshot and save the file in object shot.
* Save the file in D drive as shot1.png.

#18) moveToElement()

**moveToElement() from the Actions class to simulate mouse hover effect.**

There are scenarios where we need to hover over web elements like over menu to see submenu, links to see color changes etc. In these cases, we use Actions class. Take a look at the below syntax for Action class.

**Syntax:**

Actions actions = new Actions(driver);

WebElement mouseHover = driver.findElement(By.xpath("//div[@id='mainmenu1']/div"));

actions.moveToElement(mouseHover);

actions.perform();

**Explanation**

* Find and Locate the web element with div id ‘mainmenu1’.
* Move the mouse pointer to the element.

#19) dragAndDrop()

**dragAndDrop() from Actions class to drag an element and drop it on another element.**

In some scenarios, we might want to drag elements. **For Example,** drag an image to the stage. In this case, we can use the Actions class.

In the dragAndDrop method, we pass the two parameters, Source locator- the element we want to drag and Destination locator- the element to which we want to drop.

**Syntax:**

WebElement sourceLocator = driver.findElement(By.xpath("//\*[@id='image1']/a"));

WebElement destinationLocator = driver.findElement(By.xpath("//\*[@id='stage']/li"));

Actions actions=new Actions(driver);

actions.dragAndDrop(sourceLocator, destinationLocator).build().perform();

**Explanation:**

* Find and Locate the source web element.
* Find and Locate the destination web element.
* Drag and drop the source element on the destination element.

#20) switchTo() and accept(), dismiss() and sendKeys()

**switchTo() and accept(), dismiss() and sendKeys() methods from Alert class to switch to popup alerts and handle them.**

To switch to alerts, popups and handle them, we use a combination the of **switchTo()** and **accept(), dismiss()**methods from the Alert class.

**Syntax:**

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

alert.sendKeys("This Is Softwaretestinghelp");

alert.accept()

**Explanation:**

* Switch to the alert window.
* Type “This Is Softwaretestinghelp” inside the alert.
* Accept the alert and close it.

*alert.dismiss()* can be used to dismiss the alert.

#21) getWindowHandle() and getWindowHandles()

**getWindowHandle() and getWindowHandles() to handle Multiple Windows in Selenium WebDriver.**

There are many cases where web applications have many frames or windows.

Those are mostly advertisements or information popup windows. We can handle multiple windows using Windows Handlers. Webdriver stores a unique window id for each window. We make use of this id to handle them.

**Syntax:**

String handle= driver.getWindowHandle();

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

The above commands are used to get window ids of the current window and all the windows respectively. Please see the loop below to see how can we go to each window through for loop.

for (String handle : driver.getWindowHandles()){

driver.switchTo().window(handle);

}

**Explanation:**

* For each window handle id from driver.getWindowHandles(), switch to that window id.

#22) getConnection()

**getConnection() from DriverManager to start Database Connection.**

In order to start a database connection, we use getConnection from DriverManager class.

**Syntax:**

DriverManager.getConnection(URL, "username", "password" )

**Explanation:**

* Connect to the Database through URL and credentials.

#23) POI

**POI to read from the excel files.**

In data driven testing, we often save inputs in excel file and read it. In order to do this in WebDriver, we import POI package and then use the below command.

**Syntax:**

Workbook workbook = WorkbookFactory.create(new FileInputStream(file));

Sheet sheet = workbook.getSheetAt(0);

**Explanation:**

* Create a reader file.
* Read the file.

#24) assertEquals(),assertNotEquals(), assertTrue() and assertFalse()

**Asserts using assertEquals(),assertNotEquals(), assertTrue() and assertFalse() to compare the results.**

Assertions are used to compare the expected and actual results. Pass or fail of a test is usually decided from the result of assertions. Different types of assert are used in automation.

**Syntax:**

Assert.assertEquals(message, “This text”);

Assert.assertNotEquals(message, “This text”);

Assert.assertTrue(result<0);

Assert.assertFalse(result<0);

**Explanation:**

* In the first command, whenever the expected and actual values are same, the assertion passes with no exception. i.e., if the message is “This text”, then the assertion passes.
* In the second command, whenever the expected and actual values are same, the assertion fails with an exception. i.e., if the message is “This text”, then the assertion fails.
* In the third command, if the condition passes, assertion passes. i.e., if result<0, then the assertion passes.
* In the fourth command, if the condition passes, the assertion fails. i.e., if result<0, then the assertion fails.

#25) close() and quit()

**close() and quit() to close windows and driver instances.**

These commands are used at the end of every automation program.

**Syntax:**

driver.close()

driver.quit()

**Explanation:**

The first command closes the current window.

The second command quits this driver instance, closing every associated window, which is opened.

**Conclusion**

In this tutorial, we introduced various WebDriver’s commonly and excessively used commands. We tried to explain the commands with suitable examples and code snippets.

I have tried my best to explain the most popular WebDriver commands that we use routinely in our day to day work. These commands will let you work easily with Selenium.

I hope it was interesting and knowledgeable to you.

***Are you an Automation Tester who has tried any of the above commands? Or have we missed out any commands that you are using in the above list?***

[**Next Tutorial #18**](https://www.softwaretestinghelp.com/selenium-tutorial-18/): In the upcoming tutorial, we would discuss about **Web tables, frames and dynamic elements** which are an essential part of any web project. We will also cover the **exception handling** the important topic in more details in one of the upcoming Selenium Tutorials.

Recommended Reading

* [Cucumber Selenium Tutorial: Cucumber Java Selenium WebDriver Integration](https://www.softwaretestinghelp.com/selenium-webdriver-cucumber-selenium-tutorial-31/)
* [Introduction to Selenium WebDriver – Selenium Tutorial #8](https://www.softwaretestinghelp.com/selenium-webdriver-selenium-tutorial-8/)
* [How to Run Selenium WebDriver in Different Popular Browsers](https://www.softwaretestinghelp.com/how-to-use-different-browsers-drivers-for-your-selenium-script/)
* [Implementation of Our First WebDriver Script – Selenium WebDriver Tutorial #10](https://www.softwaretestinghelp.com/selenium-webdriver-tutorial-10/)
* [Selenium FAQs](https://www.softwaretestinghelp.com/selenium-faqs/)
* [How to Handle Alerts/Popups in Selenium WebDriver - Selenium Tutorial #16](https://www.softwaretestinghelp.com/handle-alerts-popups-selenium-webdriver-selenium-tutorial-16/)
* [GeckoDriver Selenium Tutorial: How to Use GeckoDriver in Selenium Projects](https://www.softwaretestinghelp.com/geckodriver-selenium-tutorial/)
* [Implicit and Explicit Wait in Selenium WebDriver (Types of Selenium Waits)](https://www.softwaretestinghelp.com/selenium-webdriver-waits-selenium-tutorial-15/)