**Various Commonly and Routinely Used Selenium WebDriver Commands**

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. **Selenium WebDriver Commands**:

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:**

|  |  |  |
| --- | --- | --- |
| 1 | public void explicitWaitForWinHandle(final WebDriver dvr,int timeOut, final boolean close) throws WeblivException | |
| 2 | { |

|  |  |
| --- | --- |
| 3 |  |
| 4 | try { | |

|  |  |
| --- | --- |
| 5 | Wait<WebDriver> wait = new WebDriverWait(dvr, timeOut); |
| 6 | ExpectedCondition<Boolean> condition = newExpectedCondition<Boolean>() { | |

|  |  |
| --- | --- |
| 7 |  |
| 8 | @Override | |

|  |  |  |
| --- | --- | --- |
| 9 | public Boolean apply(WebDriver d) { | |
| 10 | | int winHandleNum = d.getWindowHandles().size(); | |

|  |  |
| --- | --- |
| 11 |  |
| 12 | if (winHandleNum > 1) | |

|  |  |
| --- | --- |
| 13 | { |
| 14 | // Switch to new window opened | |

|  |  |  |
| --- | --- | --- |
| 15 | for (String winHandle : d.getWindowHandles()) | |
| 16 | { |

|  |  |  |
| --- | --- | --- |
| 17 | dvr.switchTo().window(winHandle); | |
| 18 |  |

|  |  |
| --- | --- |
| 19 | // Close the delete window as it is not needed |
| 20 | if (close && dvr.getTitle().equals("Demo Delete Window")) | |

|  |  |
| --- | --- |
| 21 | { |
| 22 | dvr.findElement(By.name("ok")).click(); | |

|  |  |
| --- | --- |
| 23 | } |
| 24 | } |

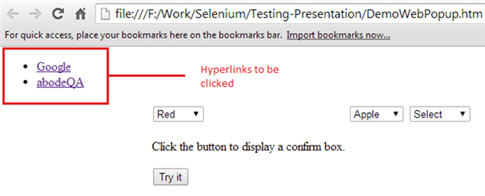
|  |  |  |
| --- | --- | --- |
| 25 | return true; | |
| 26 | } |

|  |  |  |
| --- | --- | --- |
| 27 | return false; | |
| 28 | } |

|  |  |
| --- | --- |
| 29 | }; |

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

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

[](http://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:

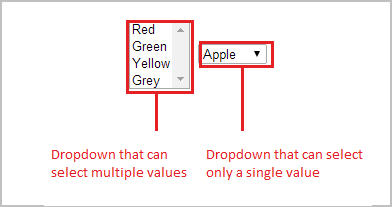
1. **Single select dropdown**: A dropdown that allows only single value to be selected at a time.
2. **Multi select dropdown**: A dropdown that allows multiple values to be selected at a time.

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

|  |  |  |
| --- | --- | --- |
| 1 | <select id="SelectID\_One" multiple=""> | |
| 2 | <option value="redvalue">Red</option> |

|  |  |
| --- | --- |
| 3 | <option value="greenvalue">Green</option> |
| 4 | <option value="yellowvalue">Yellow</option> | |

|  |  |  |
| --- | --- | --- |
| 5 | <option value="greyvalue">Grey</option> | |
| 6 | </select> |

[](http://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.

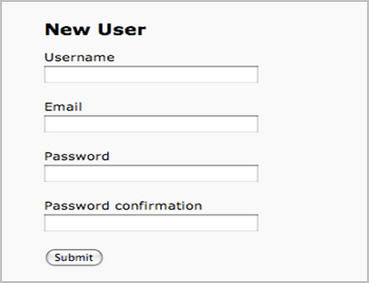
|  |  |
| --- | --- |
| 1 | // select the multiple values from a dropdown |
| 2 | Select selectByValue = newSelect(driver.findElement(By.id("SelectID\_One"))); | |

|  |  |  |
| --- | --- | --- |
| 3 | selectByValue.selectByValue("greenvalue"); | |
| 4 | selectByValue.selectByVisibleText("Red"); |

|  |  |
| --- | --- |
| 5 | 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. User may come across several types of forms like Login form, Registration form, File Upload form, Profile Creation form etc.

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

In WebDriver, 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:**

|  |  |
| --- | --- |
| 1 | // enter a valid username |
| 2 | driver.findElement(By.<em>id</em>("username")).sendKeys("name"); | |

|  |  |
| --- | --- |
| 3 |  |
| 4 | // enter a valid email address | |

|  |  |  |
| --- | --- | --- |
| 5 | driver.findElement(By.<em>id</em>("email")).sendKeys("name@abc.com"); | |
| 6 |  |

|  |  |
| --- | --- |
| 7 | // enter a valid password |
| 8 | driver.findElement(By.<em>id</em>("password")).sendKeys("namepass"); | |

|  |  |  |
| --- | --- | --- |
| 9 |  | |
| 10 | | // re-enter the password | |

|  |  |  |
| --- | --- | --- |
| 11 | driver.findElement(By.<em>id</em>("passwordConf")).sendKeys("namepass"); | |
| 12 |  |

|  |  |
| --- | --- |
| 13 | // submit the form |
| 14 | 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 with in the current HTML document or simply a web page into another web page by enabling nesting.

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

|  |  |
| --- | --- |
| 1 | <html> |
| 2 | <head><title>Software Testing Help - iframe session</title> | |

|  |  |  |
| --- | --- | --- |
| 3 | </head> | |
| 4 | <body> |

|  |  |
| --- | --- |
| 5 | <div> |
| 6 | <iframe id="ParentFrame"> | |

|  |  |
| --- | --- |
| 7 | <iframe id="ChildFrame"> |
| 8 | <input type="text" id="Username">UserID</input> | |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | <input type="text" id="Password">Password</input> | | |
| 10 | | </iframe> |

|  |  |  |
| --- | --- | --- |
| 11 | <button id="LogIn">Log In</button> | |
| 12 | </iframe> |

|  |  |
| --- | --- |
| 13 | </div> |
| 14 | </body> | |

|  |  |
| --- | --- |
| 15 | </html> |

The above HTML code illustrates the presence of an embedded iframe into another iframe. Thus, to be able to access the child iframe, user is required to navigate to the parent iframe first. After performing the required operation, 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, 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, user is leveraged to locate an iframe using 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 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

|  |  |
| --- | --- |
| 1 | try{ |
| 2 | // Protected block | |

|  |  |  |
| --- | --- | --- |
| 3 | // implement java code for automation | |
| 4 | } |

|  |  |  |
| --- | --- | --- |
| 5 | catch (ExceptionName e) | |
| 6 | { |

|  |  |  |
| --- | --- | --- |
| 7 | // catch block - Catches the exceptions generated in try block without halting the program execution | |
| 8 | } |

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**

|  |  |
| --- | --- |
| 1 | try{ |
| 2 | //Protected block | |

|  |  |
| --- | --- |
| 3 | } |
| 4 | catch (ExceptionType1 e) | |

|  |  |
| --- | --- |
| 5 | { |
| 6 | // catch block | |

|  |  |
| --- | --- |
| 7 | } |
| 8 | catch (ExceptionType2 e) | |

|  |  |  |
| --- | --- | --- |
| 9 | { | |
| 10 | | // catch block | |

|  |  |
| --- | --- |
| 11 | } |
| 12 | catch (ExceptionType3 e) | |

|  |  |
| --- | --- |
| 13 | { |
| 14 | // catch block | |

|  |  |
| --- | --- |
| 15 | } |

In the above code, 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:

|  |  |  |
| --- | --- | --- |
| 1 | WebElement saveButton = driver.findElement(By.id("Save")); | |
| 2 | try{ |

|  |  |  |
| --- | --- | --- |
| 3 | if(saveButton.isDisplayed()){ | |
| 4 | saveButton.click(); |

|  |  |  |
| --- | --- | --- |
| 5 | } | |
| 6 | } |

|  |  |  |
| --- | --- | --- |
| 7 | catch(NoSuchElementException e){ | |
| 8 | e.printStackTrace(); |

|  |  |
| --- | --- |
| 9 | } |

### ****Conclusion****

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