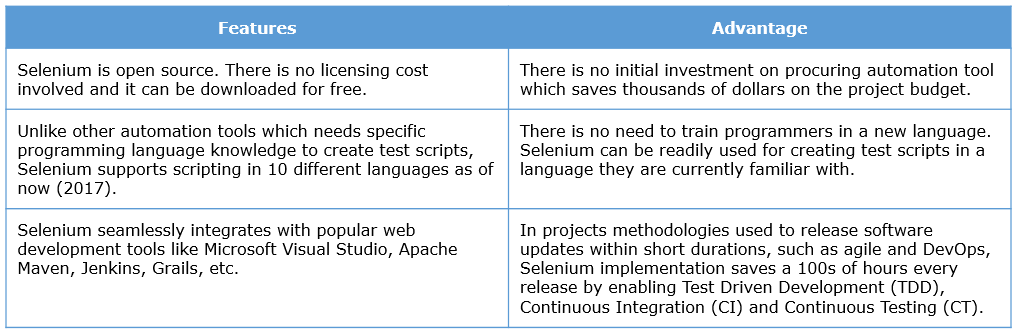
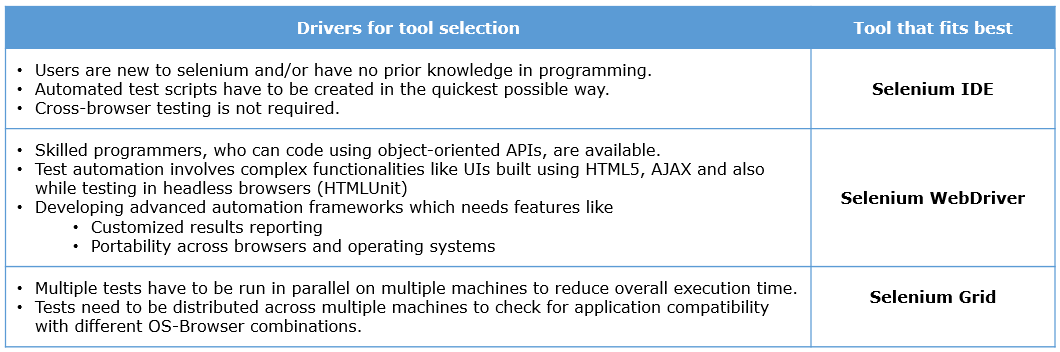
**Selenium Feature and Advantage**



Selenium is a suite of tools which can automate actions on web browsers. Each tool is developed to support a specific kind of automation approach.

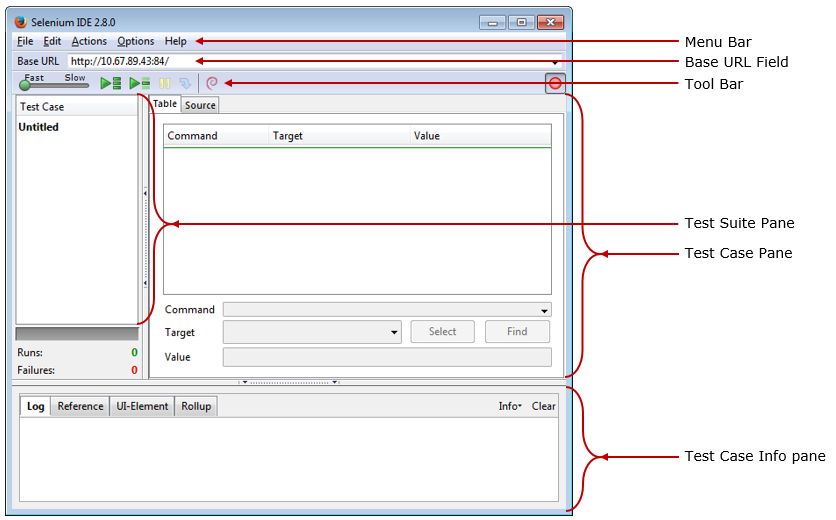
The three important tools that form the core of selenium ecosystem are

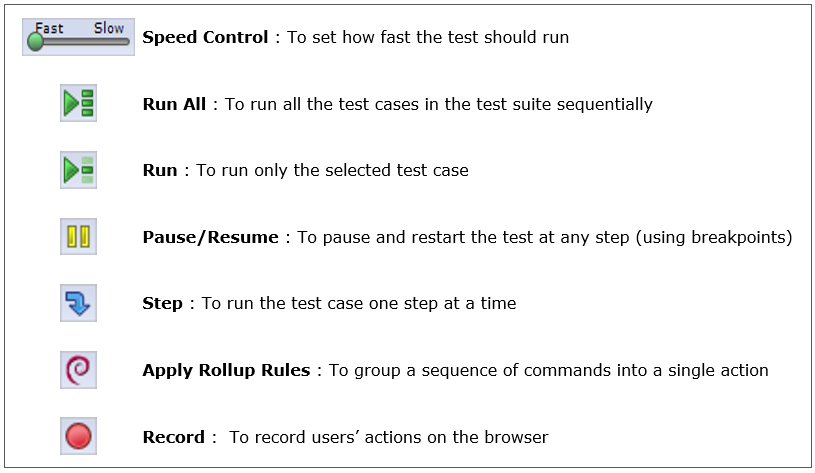


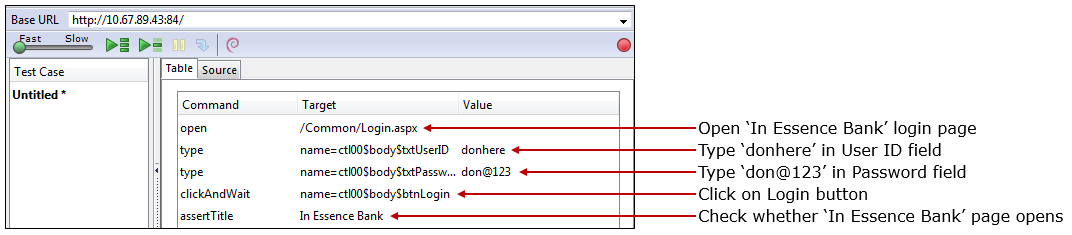
**What is Selenium IDE**

Selenium Integrated Development Environment (IDE) is a tool that you can use to record, edit, debug and playback test Selenium test scripts.

* It is an easy-to-use Mozilla Firefox browser Add-on.
* Even if you don't know any programming language, you can easily create automation test scripts in Selenium IDE by recording user actions.
* Working with Selenium IDE is an effective method for you to learn how Selenium works.







There are two main parts of the test step.

**Command**

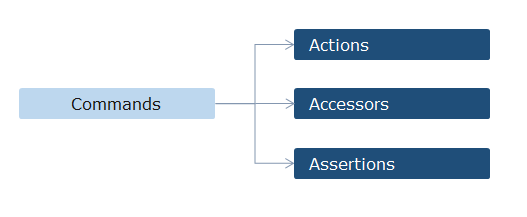
* Commands are predefined keywords which tell selenium what to do.
* Selenium provides a rich set of commands for fully testing your web application in virtually any way you can imagine
* This command set, which essentially represents the vocabulary used in selenium, is called ‘Selenese’.
* A test script is a sequence of ‘Selenese’ commands, one in each test step.

**Parameters**

* Depending on the ‘Selenese’ command’s syntax, parameters may or may not be passed to them. When parameters are needed it may be passed using either one or both the fields as described below.
* **Target :** This field often contains information that uniquely identifies the element, in the Application Under Test (AUT), on which the command needs to be executed.
* **Value :** The value field is usually filled with a text pattern or a selenium variable name. Its value is used to aid the command to perform its action on the target element. For example, a text to be entered into the field.

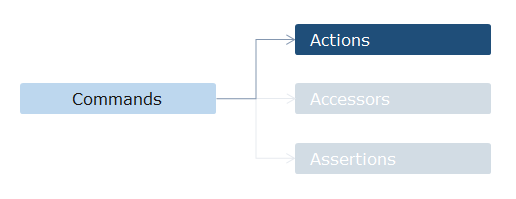
**Selenese Command**

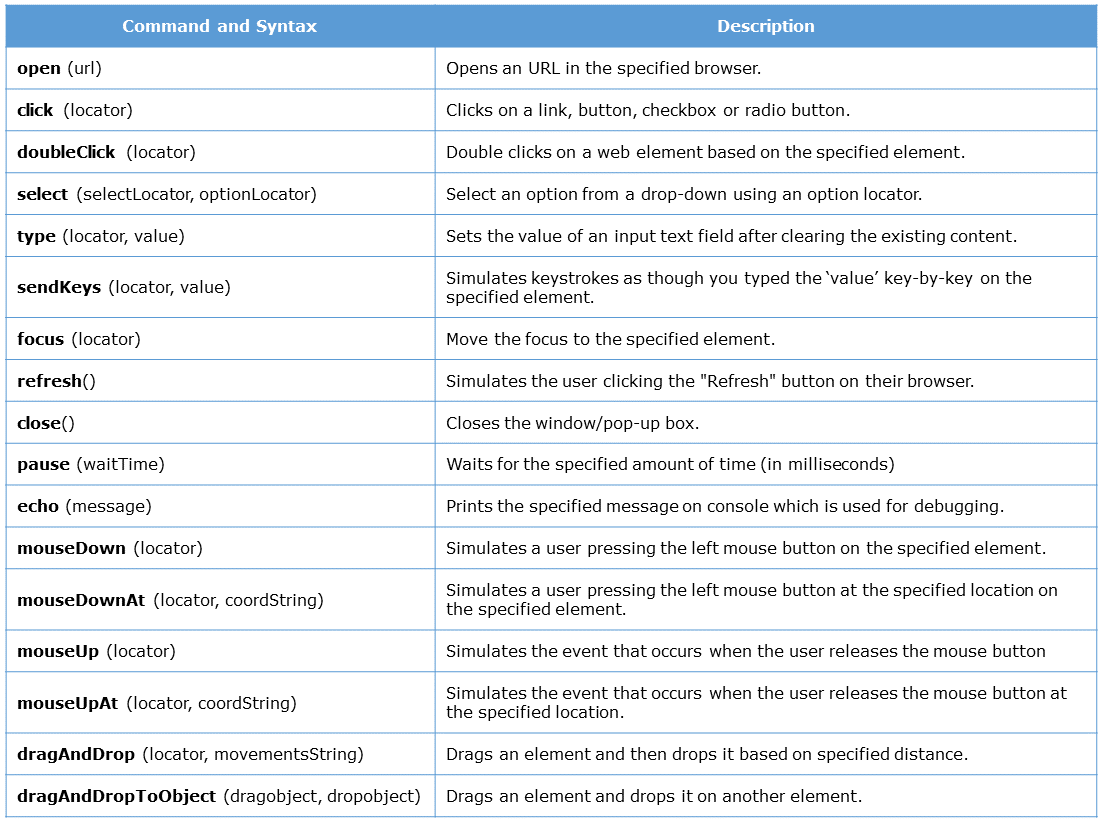
There are over 500 selenese commands that are available at your disposal. These commands can be classified into the following 3 categories.



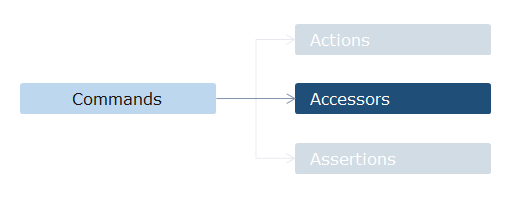
Let’s look at each one of them in detail.

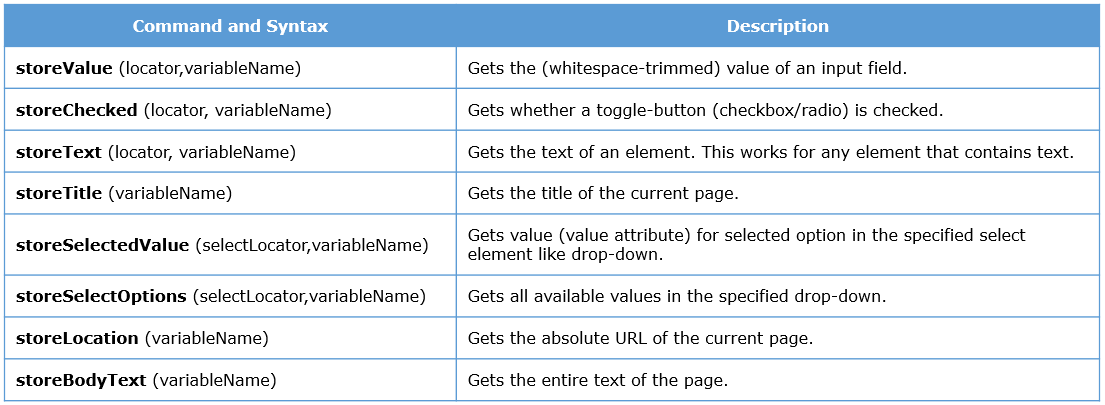
**Actions** are commands that you can use to perform an action on the application in situations like populating an edit box, selecting a drop-down menu value, clicking on a link, etc.



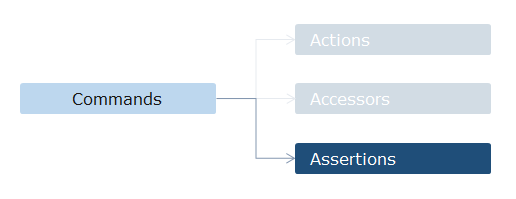


**Accessors** are commands that you can use to extract values from the application in situations like getting the URL of a page opened by the application, getting the customer number generated by the application, etc.





**Assertions** are commands used to validate actual values in the application against expected values in situations like checking if the page title is xxx, checking whether a success message is displayed at the end of creating an account, etc.



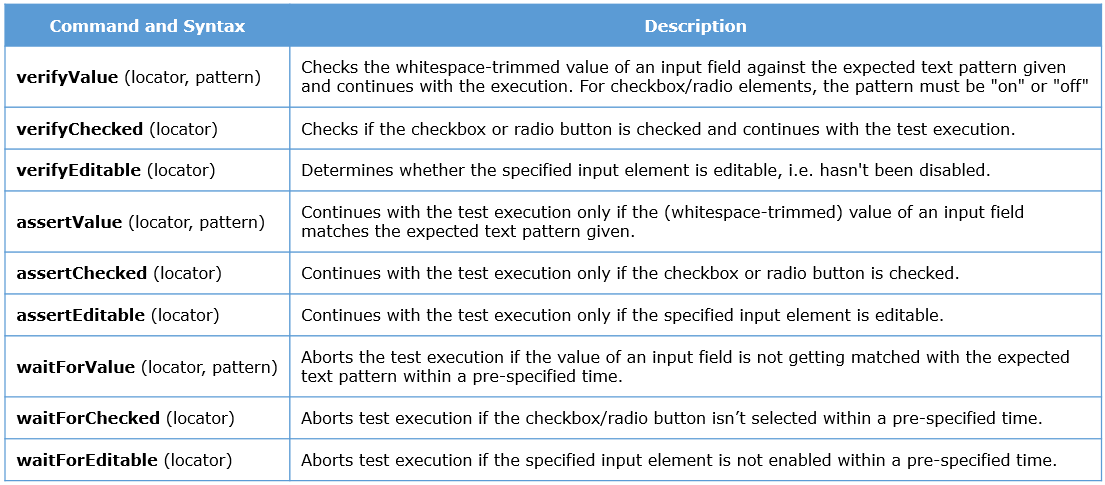
They verify whether the state of the application conforms to what is expected at any required point in time.

They come in three flavors:-

**Verify:** Will immediately perform the check and even if the check fails, it continues the test script execution.

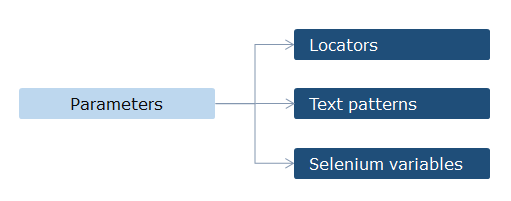
**Assert:** Will immediately perform the check and if the check fails, it aborts the test script execution.

**WaitFor:** Instead of performing the check immediately it wait for the check to pass or until a pre-specified time, whichever happens first. If the check fails even after the pre-specified time, it aborts the test

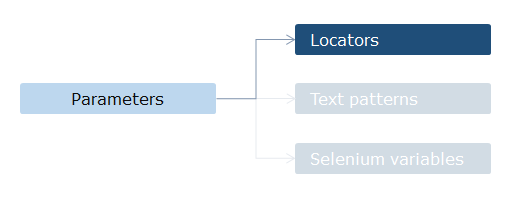


**Types of command parameters**

The parameters that you need to pass to a selenese command, as required by it's syntax, can be classified into 3 types.

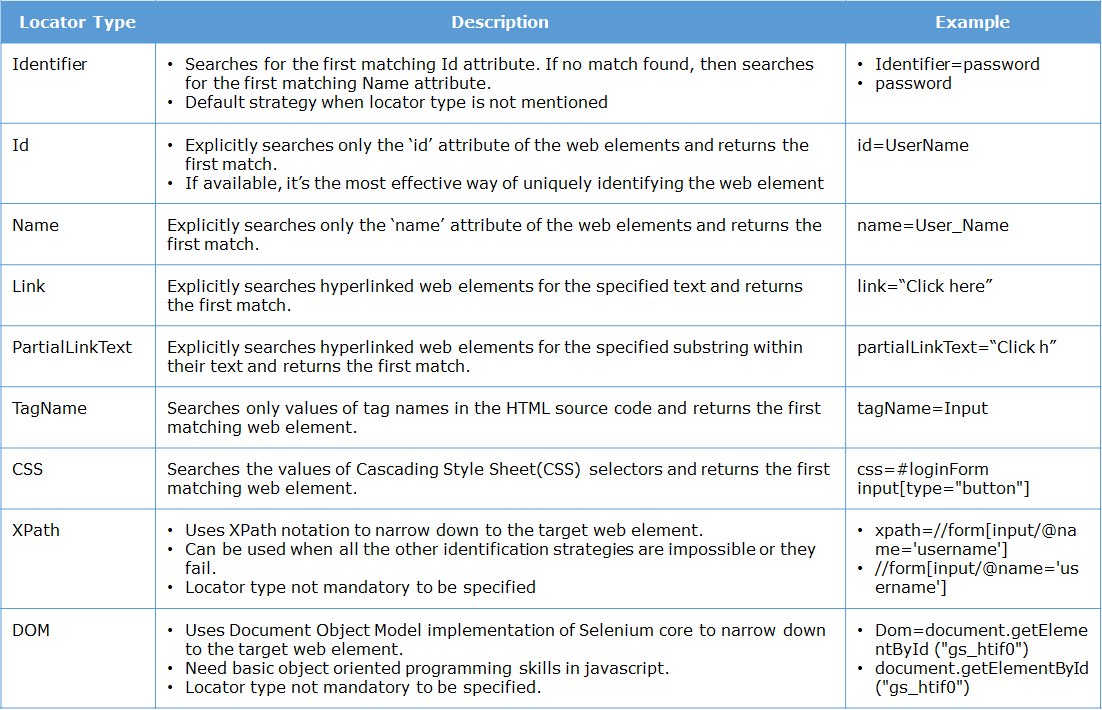


A **locator** is an expression that tells Selenium which HTML element of the application a command has to perform the action on.

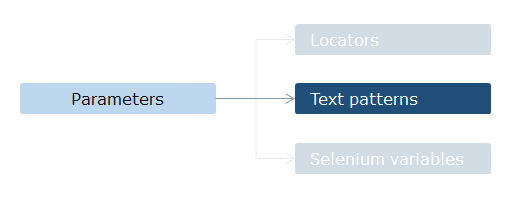


Every field/object/control available on the web page UI is called a web element. Selenium identifies a web element in a web page by parsing through its HTML source code.

The locator types available in selenium are

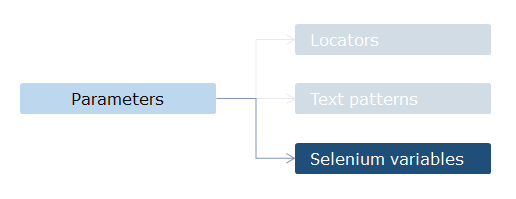


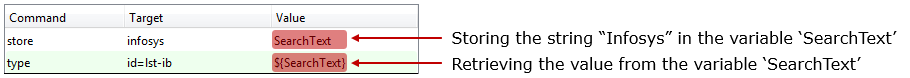
A **Text Pattern** is an expression which represents a value that needs to be used by the command while performing its operation.





You can use **Selenium variables** to store text expressions as strings which can be reused later as locator expressions or text patterns.





**Common customization techniques**

There are times when a test script that you recorded fails during playback. Below are a few examples of such scenarios you might encounter while working with Selenium IDE.

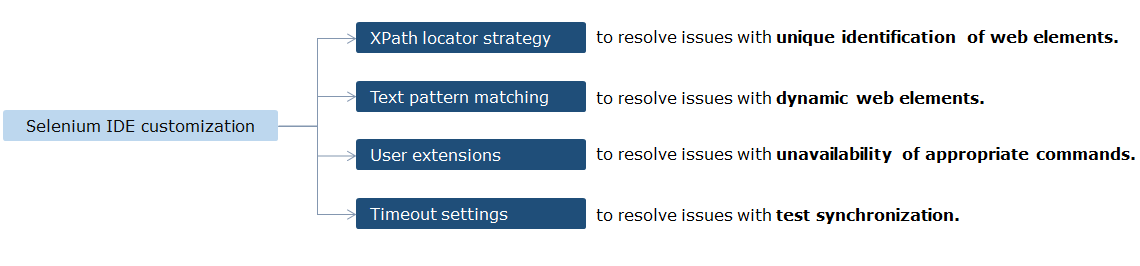
**Scenario 1** In the below test step, which is trying click a link called ‘click here’, the locator might have been used to identify a single link on the web page while recording.

**Scenario 2** In an online shopping application, If we want to test an order placement functionality, we would validate the test by checking whether an order number is generated at the end of the test using an assertion test step like the one shown below.

**Scenario 3** If you are testing an email account creation page then the below step, where you are trying to create a new user id, might fail.

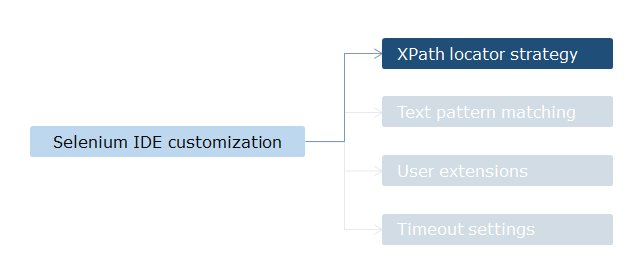
**Scenario 4** If the network speed is slow during test playback, then the pages might take long time to load. Hence when the script is trying to execute any action step on a web element, it might not be available. This kind of failures in synchronization between test script and AUT would result in the test run failing.

Let’s look at ways in which we can customize Selenium IDE tests to resolve issues such as the ones listed above.



**What is XPath**

XPath is short form for XML Path. It is a text expression syntax which you can use for selecting parts of a document written in XML (or XML based mark-up language like HTML).



**How XPath works**

Xpath is a machine readable way in which you can represent the path to reach a required node starting from a specific node (termed as context node) through other nodes.

There are 2 types of XPath:-

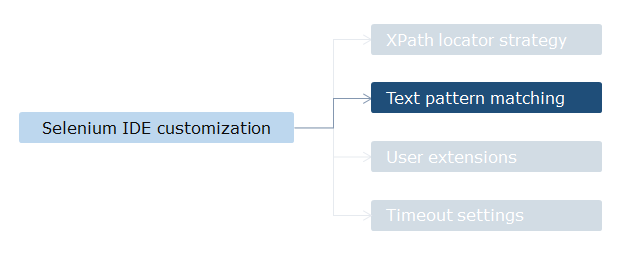
**Absolute XPath**

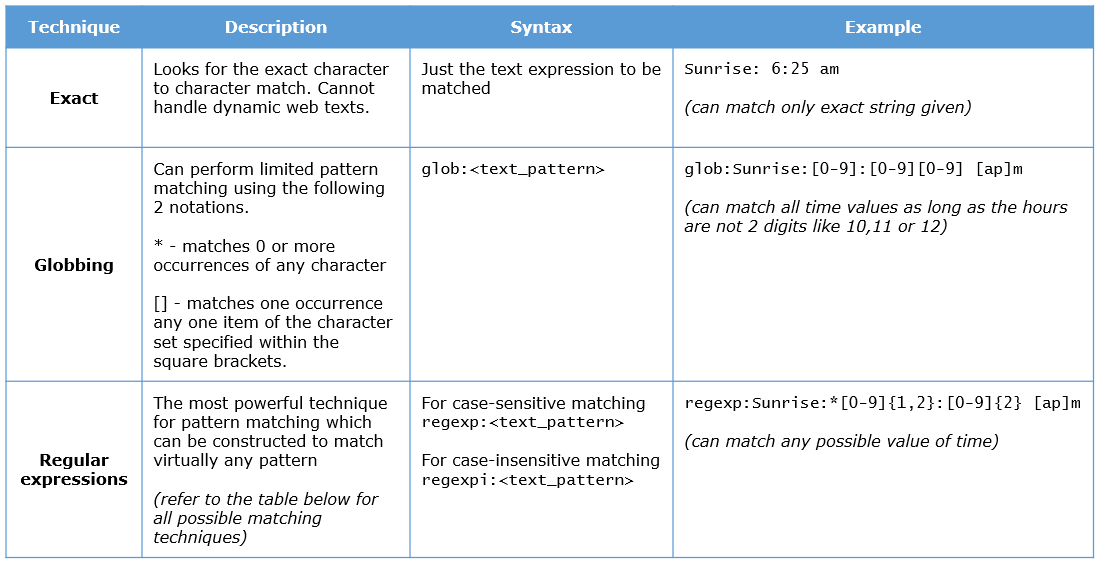
* The path starts from the root node.
* The expression starts with a forward slash (/).
* The advantage of using absolute XPath is that it identifies the element very fast.
* Disadvantage of absolute XPath is that if there any change in the xml/html document like some other tag added in between, then this path will no longer works.

**Relative Xpath**

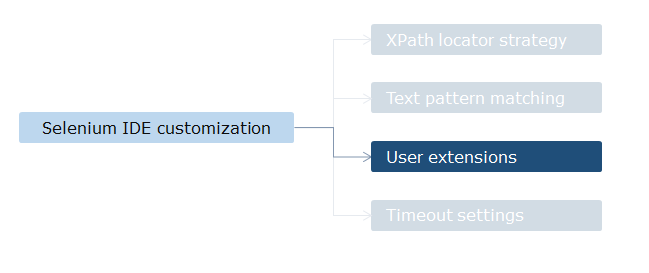
* The path starts from the node of your choice (other than the root node).
* The expression starts with Double forward slash(//).
* Advantage of using relative XPath is that you don't need to mention a long XPath expression. you can start from the middle or in between.
* Disadvantage of using relative XPath is that it will take more time in identifying the element as we specify the partial path and not exact path.
* If there are multiple nodes for the same relative path, it will return the first node that is identified.

**Pattern matching in Selenium**

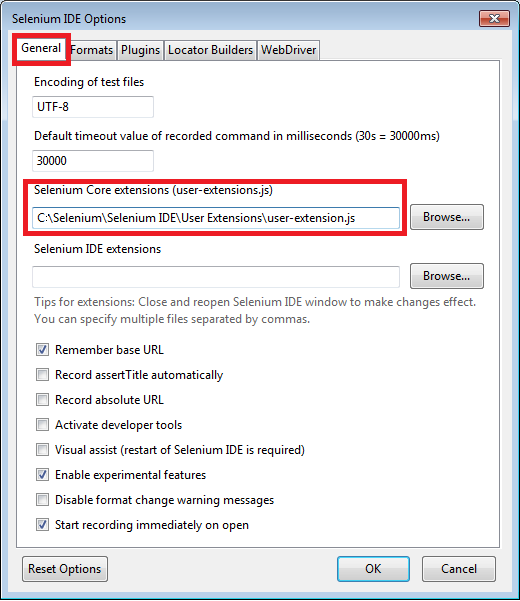




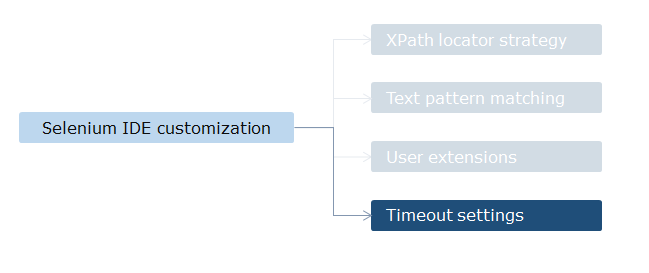
**User extensions**



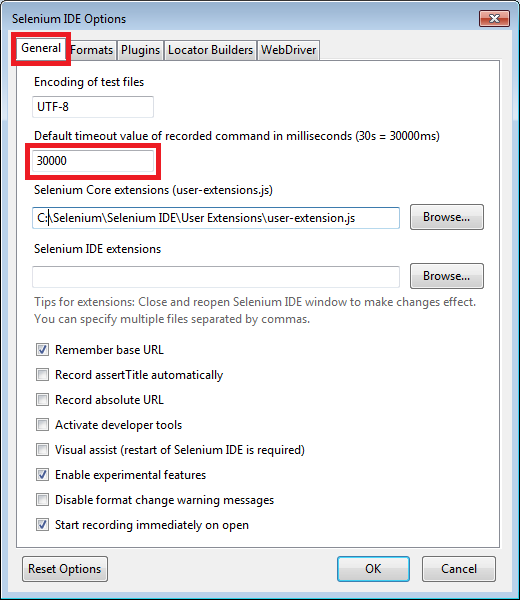
Then you can make those commands available in Selenium IDE by linking your '.js' file at 'Selenium Core extensions' settings in the 'Selenium IDE Options' dialog box. (Selenium IDE->Options Menu->Options)



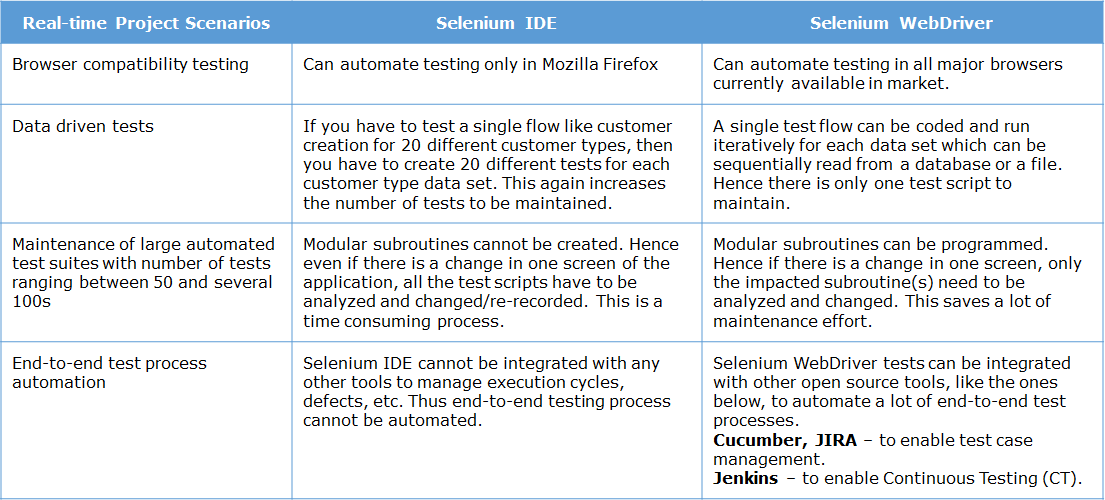
**Configuring timeout settings**



Whenever your tests fail because of web pages getting loaded slower than normal, you can make Selenium IDE to wait a bit longer until all web elements all loaded.



**Why learn Selenium WebDriver**

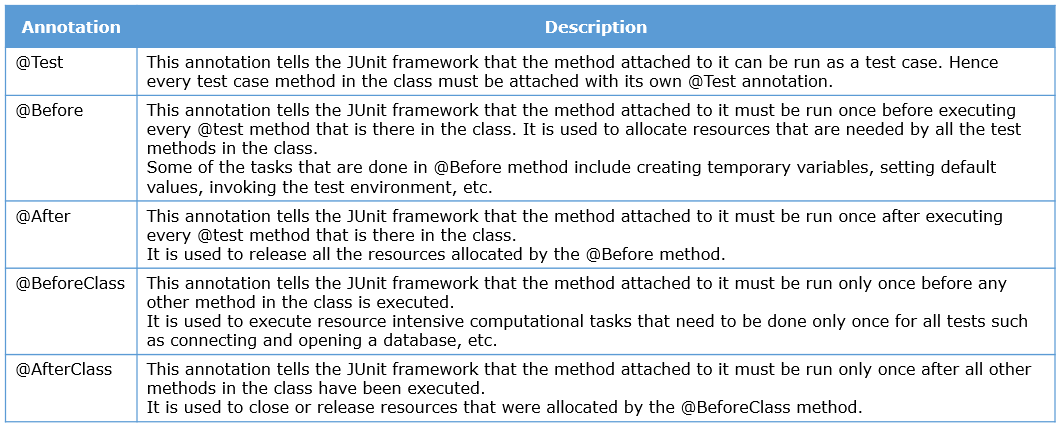


While Selenium IDE is User Interface (UI) based solution for recording and playing back tests on web pages, Selenium WebDriver is an Application Programming Interface (API) based solution for programming and executing tests on web pages.

**Features of Selenium WebDriver**

* You can run tests on any of these browsers - Internet Explorer, Firefox, Chrome, Safari and Opera.
* You can create tests using any of these programming languages – Java, C#, JavaScript, Perl, PHP, Python, R and Ruby.
* You can interact with complex UI elements like Ajax controls
* Using any freely available open source unit testing frameworks, you can create easily maintainable large test suites for your test automation project

**Working with JUnit framework**

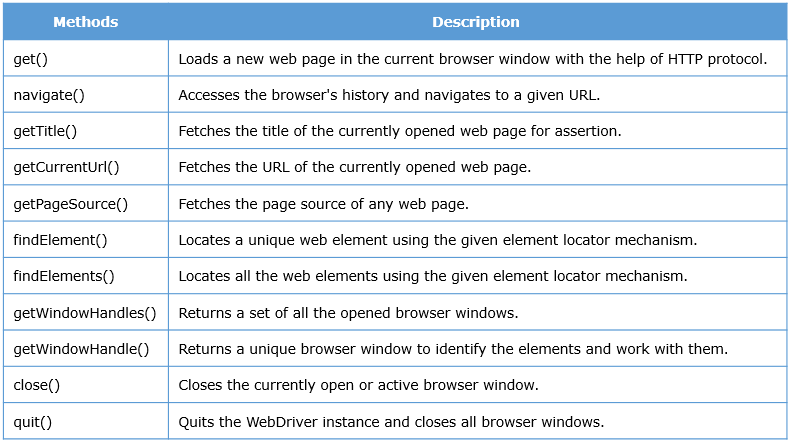


**The WebDriver interface**

The WebDriver API is the main interface you need to use for writing a test script. It represents the web browser. You can use the methods in this interface for automating actions on the browser.

It is present in the ‘org.openqa.selenium.WebDriver’ package.

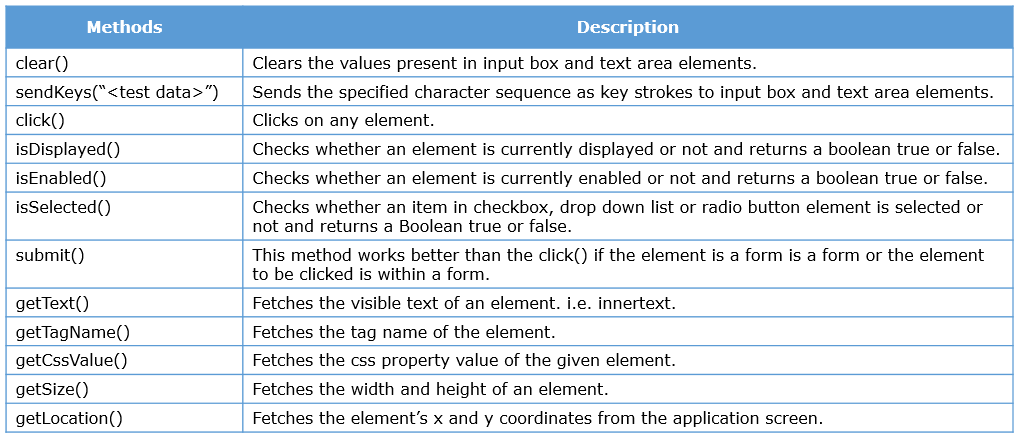
Some of the key methods, which you would be using in any selenium test, are listed below.



**The WebElement interface**

All individual items on a web page are coded as HTML elements in the HTML source code document. The WebElement interface in Selenium WebDriver is used to represent these HTML elements.

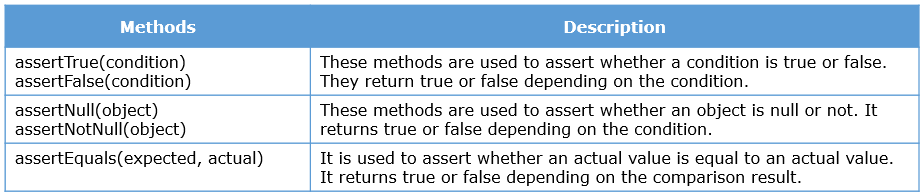
You can code automated actions on the fields (HTML elements) of the web page by using the methods of the WebElement interface.



**Assertions in JUnit framework**

The final step needed to complete our test flow is adding assertions – steps that validate whether web elements in the page are behaving as expected.

Here are the most commonly used assert methods.



**Syntax of Xpath and CSS**

**xpath**

* //tagname[@attribute='value']
* //tagname[contains(@attribute,'value']

**Site for below examples: http://www.qaclickacademy.com/interview.php**

**Traversing from One Sibling to other next siblings**

Eg: //li[@id='tablist1-tab1']/following-sibling::li[2]

**Traversing from one sibling to other previous sibling**

Eg: //li[@id='tablist1-tab2']/preceding-sibling::li[1]

**Traversing from child to parent tag**

Eg: //li[@id='tablist1-tab2']/parent::ul

**Traversing from Parent to child tag**

Eg: //ul[@class='responsive-tabs\_\_list']/child::li[2]

**Finding xpath through visible txt**

Eg: //\*[text()=' Soap UI ']

**Finding xpath starts with some Attribute**

Eg: //input[starts-with(@name,'firstn')]

**Finding xpath Ends with some Attribute**

Eg: //input[ends-with(@name,'name')]

**Finding xpath contains some Attribute**

Eg: "//input[contains(@name,'\_email\_')]"

**CSS**

* *tagname[attribute='value']*
* *tagname#id*
* *tagname.classname*
* *tagname[attribute\*='value']*

**Synchronization**

For an automation script to run without any errors due to missing or disabled elements, consecutive test step executions must wait until AUT is ready to receive the instruction from the test step. Otherwise your next test step would fail with exceptions like

NoSuchElementException

ElementNotVisibleException

**Implicit wait**

In Selenium IDE, we achieved test synchronization using the timeout settings. In Selenium WebDriver, the equivalent method is called ImplictlyWait().

Usage:

***driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);***

This method accepts 2 parameters, the first parameter will accept the time as an integer value and the second parameter will accept the time measurement in terms of SECONDS, MINUTES, MILISECOND, MICROSECONDS, NANOSECONDS, DAYS, HOURS, etc.

The drawback with implicit wait is that it is applied to all the steps in the script. Hence it’s value has to be configured for the Web element with the longest wait time required.

**Explicit wait**

While you can use implicit wait as a generic timeout setting, you can use explicit wait to wait only for a specific element based on a condition

Usage:

***WebDriverWait wait = new WebDriverWait(driver, 10);***

***wait.until(ExpectedConditions.elementToBeClickable(By.id("someid")));***

A new explicit wait object called 'wait' of class WebDriverWait is created which is configured to wait for a maximum duration of 10 seconds.

The 'until' method of the 'wait' object is configured to wait until the element identified by id=someid is enabled to accept a mouse click.

ExpectedConditions is a class which contains predefined common conditions that are frequently encountered when automating web browsers (like 'elementToBeClickable' in this example). For a complete list of conditions in the ExpectedConditions class please refer to the appendix.

**Fluent Wait:**

The fluent wait is used to tell the web driver to wait for a condition, as well as the **frequency** with which we want to check the condition before throwing "ElementNotVisibleException" exception.

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

***.withTimeout(30, TimeUnit.SECONDS)***

***.pollingEvery(5, TimeUnit.SECONDS)***

***.ignoring(NoSuchElementException.class);***

In the above example, we are declaring a fluent wait with the timeout of 30 seconds and the freque-ncy is set to 5 seconds by ignoring "**NoSuchElementException**"

**Automate check boxes, radio buttons and drop down lists**

Usage Example:

*//Find the My Deposits dropdown list and store it as a WebElement*

*WebElement ddlDeposit = driver.findElement(By.id("ctl00\_ctl00\_body\_cph\_Deposit\_ddlMyDeposits"));*

*//Pass the reference variable for ddlDeposit as a parameter for the Select class*

*Select select = new Select(ddlDeposit);*

*//Note: Values in dropdown list can be accessed using either index number or value or visible text.*

*//Note: Index number starts from 'Zero'*

*//Use the select reference variable for selecting any option using index value approach*

*select.selectByIndex(3);*

*//Select the Change in Transfer mode checkbox*

*driver.findElement(By.id("ctl00\_ctl00\_body\_cph\_Deposit\_chkNewTransfer")).click();*

**Web tables**

Web tables are objects on a web page which are used to present data in a tabular format

//Fetch the Account details table

WebElement tblAccountDetails = driver.findElement(By.id("ctl00\_ctl00\_body\_cph\_MyAccount\_gvAccountDetails"));

/Fetch all the rows inside the Account details table using the tr tag and store it in rows list

List<WebElement> rows = tblAccountDetails.findElements(By.tagName("tr"))

//Iterate over all the rows

for (WebElement row : rows) {

//Find all the cols inside the particular row using the td tag

List<WebElement> cols = row.findElements(By.tagName("td"));

//Iterate over all the columns inside the particular row

for (WebElement col : cols) {

//Print the cell value

System.out.println("Column value: " + col.getText());

}

}

**Frames**

In the world of web applications, a frame is a part of the web page which has the ability to load and display content independent of the container page in which it is in.

How frames work

* This effect of 'page within a page' is achieved by building each section as a separate HTML file and having one "master" HTML file identify all of the sections.
* When a user requests a web page that uses frames, the address requested is actually that of the "master" file that defines the frames.
* The result of the request is that multiple HTML files are returned, one for each visual section.
* In the HTML master file, they are defined with FRAMESET and FRAME tags.

//Switch to the frame using the index number

driver.switchTo().frame(1);

//Switch back to the parent frame

driver.switchTo().parentFrame()

//Try to switch to another frame using frame name

driver.switchTo().frame("right");

**Automate interaction with multiple browser tabs**

// Fetch the number of opened windows

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

System.out.println("Number of opened windows: " + windowHandles.size());

//Iterate through all the available windows

for (String string : windowHandles) {

//Switch between windows using the string reference variable

driver.switchTo().window(string);

//Fetch the url of the page post successful switch

String currentUrl = driver.getCurrentUrl();

//check whether the url post switch is the desired page

if (currentUrl.equals("http://10.67.89.43:84/Common/ContactUs.aspx")) {

temp = 1;

break;

} else {

temp = 0;

}

}

if (temp == 1)

System.out.println("Window found.");

else if (temp == 0)

System.out.println("Desired Window not found.");

**Pop-up windows**

Message pop-ups: Displays a message from the web application. Usually has only OK and CANCEL buttons.

Prompts: Expects an input (usually text) from the user.

Confirmations: Asks for user’s confirmation to do a particular action. Can have multiple buttons such as ACCEPT, REJECT, YES, NO, ABORT, RETRY, etc.

//Using the driver reference variable switch to the alert box and click on OK button using accept method

driver.switchTo().alert().accept();

//Using the driver reference variable switch to the alert box and send some values

driver.switchTo().alert().sendKeys("Peter Parker");

**Drag and drops**

In web pages which are written in HTML 5, you can ‘grab’ any element on the screen, ‘drag’ it to some other position on the page and, if needed, ‘drop’ it on another element. This is called drag and drop feature.

*//Fetch the element property which should be dragged*

*WebElement fromElement = driver.findElement(By.id("draggable"));*

*//Fetch the element property where the dragged element should get released*

*WebElement toElement = driver.findElement(By.id("droppable"));*

*//Create a reference for Actions class*

*Actions action = new Actions(driver);*

*//Use dragAndDrop method and provide arguments as the from element and to element*

*action.dragAndDrop(fromElement, toElement).perform();*

**AJAX controls**

AJAX (Asynchronous JavaScript And XML) is a web development technique which uses a client-side JavaScript that

* Communicates to and from a server
* Updates parts of a web page using the information received from the server
* Without reloading the entire page.

AJAX calls are tricky scenarios for test automation as there is no page refresh for the automation to predict whether it the AJAX element is ready to be used.

In Selenium, these elements can be handled using explicit wait technique as shown in the next page.

*//Employee link*

*WebElement linkEmployee = driver.findElement(By.xpath("html/body/form/div[6]/div/div[1]/div[1]/ul/li[4]/a"));*

*//Details links --- After hovering Employee link*

*WebElement linkDetails = driver.findElement(By.xpath("html/body/form/div[6]/div/div[1]/div[1]/ul/li[4]/ul/li[1]/a"));*

*//Use the action class to*

*Actions action = new Actions(driver);*

*action.moveToElement(linkEmployee).moveToElement(linkDetails).click().build().perform();*

*//Explicit wait with a maximum of 20seconds*

*WebDriverWait wait = new WebDriverWait(driver, 20);*

*//Wait until the table gets displayed wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.id("cphMainContent\_tcEmployeeDetails\_tpUpdateQualifications\_grdQualificationForAdmin"))));*

**Dynamic web elements**

If you have visited pages of e-commerce websites, you would have noticed that the home page does not contain the same information everyday. It's content, therefore, its web elements and their properties keep changing. Such elements in a web page are called dynamic web elements.

Some dynamic web pages are designed in such a way that every time the page is reloaded, the web element properties change. Think about a news site where they have to keep updating their content minute by minute.

*//Create an explicit wait for a maximum of 20seconds*

*WebDriverWait wait=new WebDriverWait(driver, 20);*

*//Wait for the label corresponding to Infosys Pune to get visible*

*wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("lblMessage5")));*

*//Once the element is visible fetch the text and print it in the console*

*String lblDC = driver.findElement(By.id("lblMessage5")).getText();*

*System.out.println("DC Name: " + lblDC);*

**Question:** Lara is automating a web application. There are many hidden elements in the source code of the webpage. What command does she need to use to perform a click operation.

**Ans:** (JavascriptExecutor(driver)).executeScript("document.getElementsByClassName(ElementLocator).click()");

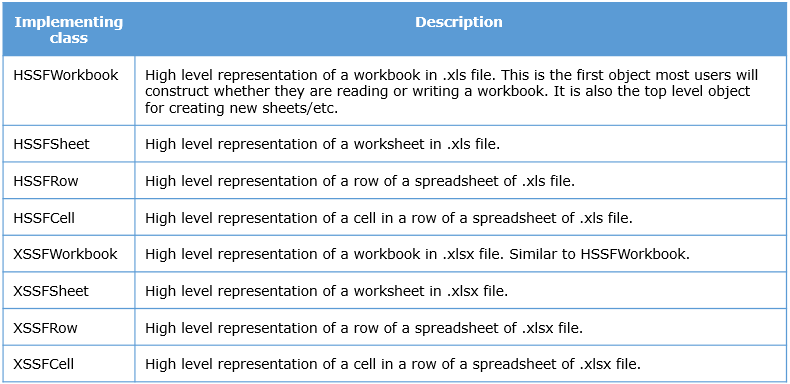
**Question:** What is the difference between executescript() & executeAsyncScript() methods of Interface JavascriptExecutor

**Ans:** executeAsyncScript() must explicitly signal that they are finished by invoking the provided callback as a parameter in the method signature where as in executescript() the script fragment provided will be executed as the body of an anonymous function

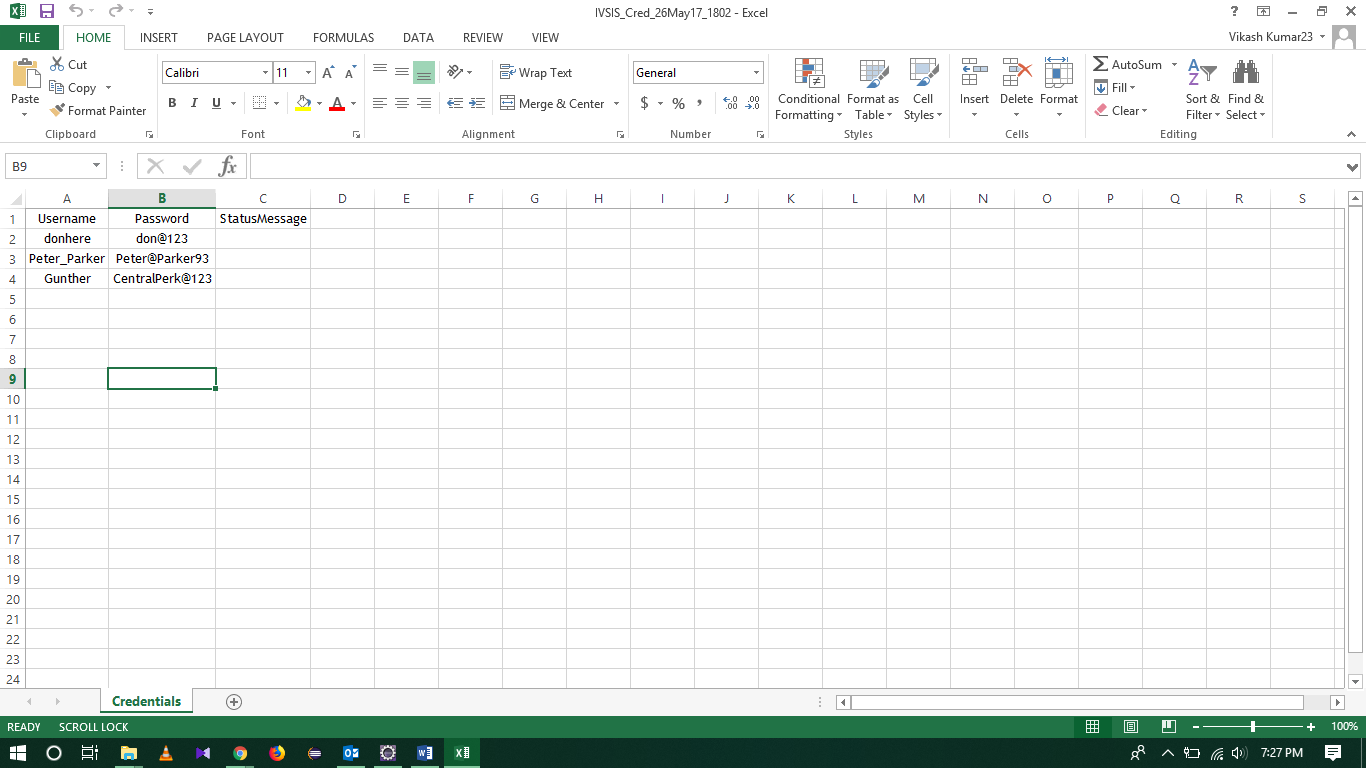
**Parameterization**

In order to implement parameterization, you need to know how to integrate excel workbooks with your selenium web driver script. The POI interface, will help you do that.

Apache POI is a popular API that allows java programmers to create, read and edit Microsoft Office files.



**Read the Excel cell data (File Input Stream)**



*//Path from where the excel file has to be read*

*String filePath = System.getProperty("user.dir")*

*//File input stream which needs the input as the file location*

*FileInputStream fileStream = new FileInputStream(filePath);*

*//Workbook reference of the excel file*

*XSSFWorkbook workbook = new XSSFWorkbook(fileStream);*

*//Sheet which needs to be accessed from within the workbook*

*XSSFSheet sheet = workbook.getSheet("Credentials");*

*//Count the number of rows*

*int rowCount = sheet.getLastRowNum() - sheet.getFirstRowNum();*

*//Iterate the Username/Password/*

*for (int i = 1; i <= rowCount; i++) {*

*//Pass the row number and the cell number from where the value has to be fetched*

*driver.findElement(By.id("ctl00\_body\_txtUserID")).sendKeys(sheet.getRow(i).getCell(0).getStringCellValue());*

*driver.findElement(By.name("ctl00$body$txtPassword")).sendKeys(sheet.getRow(i).getCell(1).getStringCellValue());*

*driver.findElement(By.id("ctl00\_body\_btnLogin")).click();*

*String welcomeMessage = driver.findElement(By.xpath("//\*[@id='ctl00\_ctl00\_divWelcome']")).getText();*

*System.out.println("Welcome message: " + welcomeMessage);*

*driver.findElement(By.linkText("Log Out")).click();*

*}*

**Write to the Excel Cell (File Output Stream)**

After successful login through above script write the welcome message to the cell

String filePath = System.getProperty("user.dir") + "\\Cred.xlsx";

//FileInputStream

//Access the workbook

//Access the sheet

for (int i = 1; i <= rowCount; i++) {

//Login and fetch the welcome message

//Get the current row where the data has to be written

Row newRow = getSheet.getRow(i);

//Create a new cell with reference to the row

Cell cell = newRow.createCell(2);

//Set the value in the cell

cell.setCellValue(welcomeMessage);

//Logout from the application.

}

//Create an output stream with the location where the file has to be created

FileOutputStream fileOutputStream = new FileOutputStream(filePath);

//Write the workbook

workbook.write(fileOutputStream);

//Close the workbook

workbook.close();

}

**Screenshot capture**

Taking screenshots of the application under test would help you in the following scenarios.

* Documenting test results.
* Providing proofs for test execution and test failures during audits and reviews.
* Helping developers while debugging defects by showing application behavior.

//Typecast the driver reference variable with TakesScreenshot for access the methods from TakesScreenshot interface

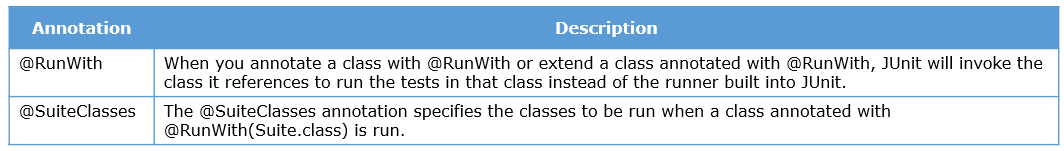
//getScreenshotAs method will take arguement for the output type of the file

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

//Using the FileUtils class copy the generated screenshot file to any location

FileUtils.copyFile(scrFile, new File("C:\\Users\\sandeep\_raulo\\Desktop\\Image.png"));

**JUnit test suite**



//RunWith annotaion

@RunWith(Suite.class)

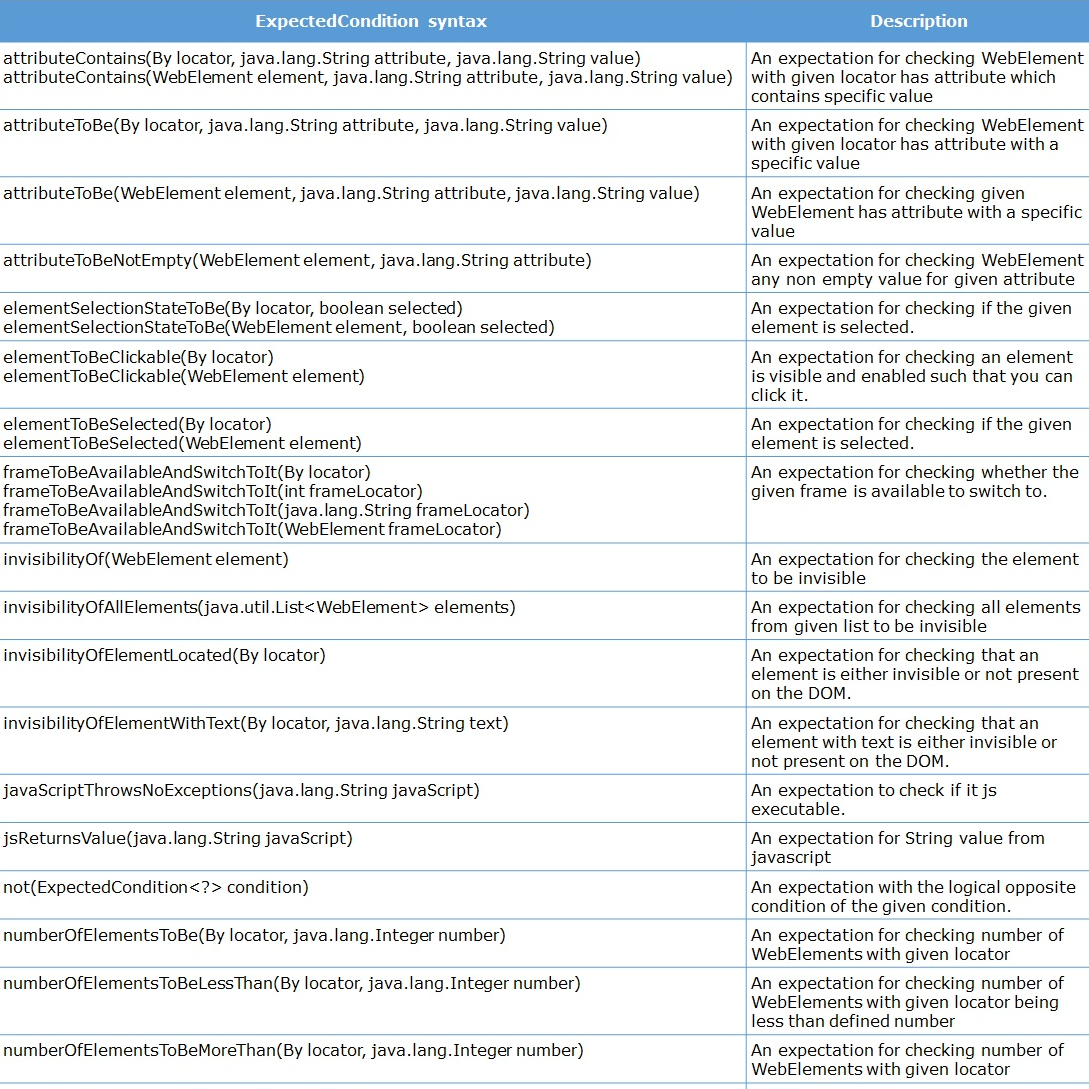
//@SuiteClass with the .class file name which needs to run as a suite

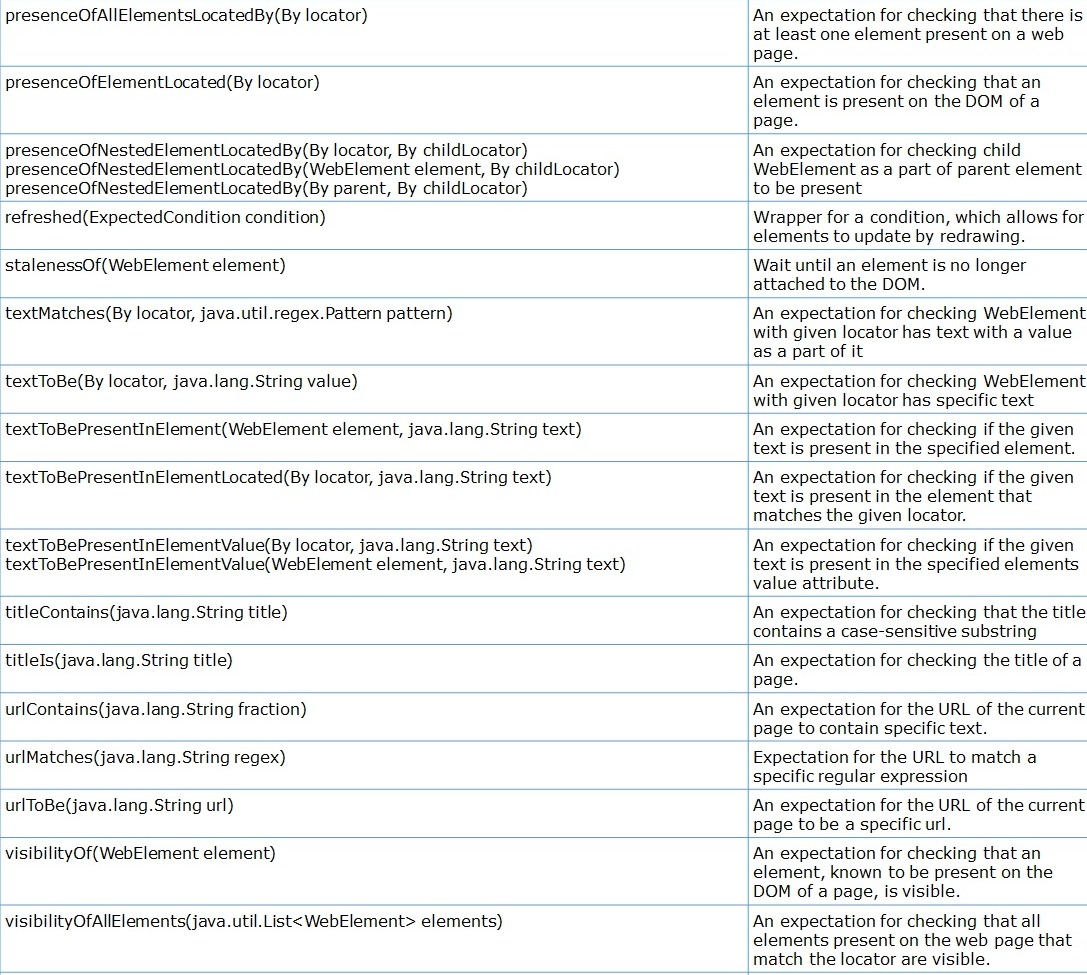
@SuiteClasses({ Test01.class, Test02.class, Test03.class })

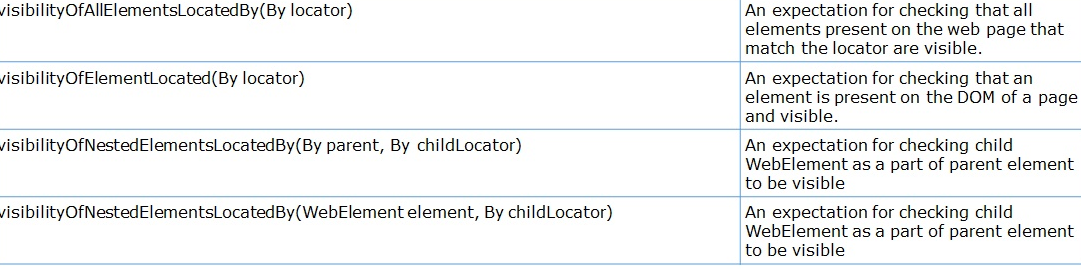
public class Test04 {

}

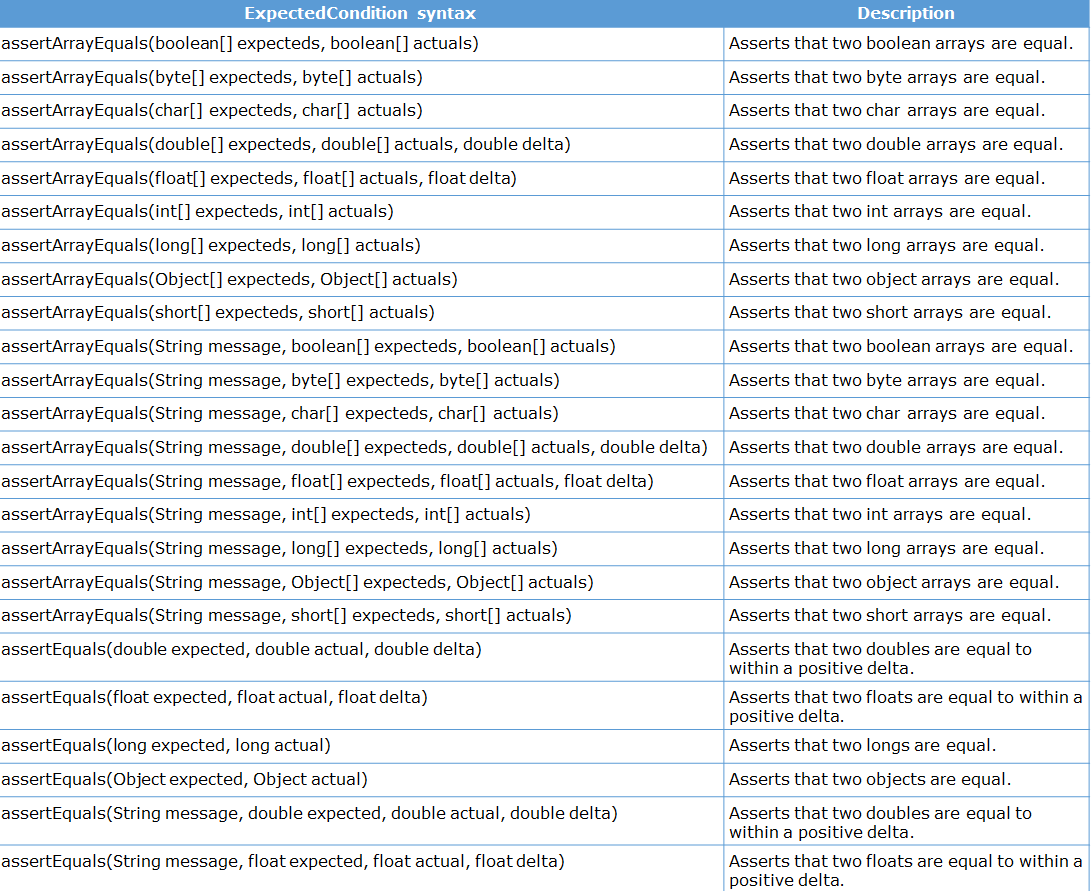
**Methods of ExpectedConditions class that you can use to specify the conditional waits while using explicit wait technique in Selenium WebDriver scripts.**

****

****

****

**Given below is the complete list of assert methods available to you in JUnit framework.**

****

****