WEBDRIVER SESSION

SELENIUM:

**Selenium is a free (open source) automated testing suite for web applications across different browsers and platforms.** It is quite similar to HP Quick Test Pro (QTP) only that Selenium focuses on automating web-based applications.

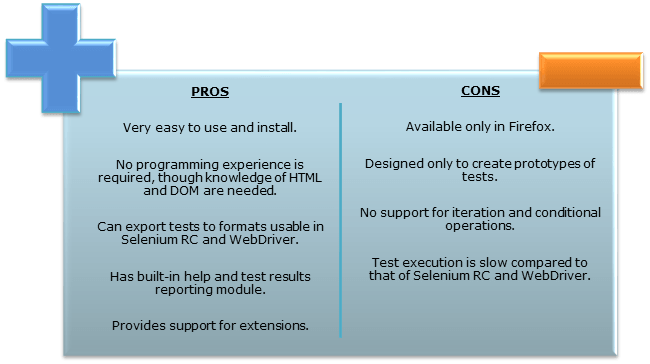
Selenium is not just a single tool but a suite of software's, each catering to different testing needs of an organization. **It has four components.**

* Selenium Integrated Development Environment (IDE)
* Selenium Remote Control (RC)
* WebDriver
* Selenium Grid

Selenium RC and WebDriver are merged into a single framework to form **Selenium 2**. Selenium 1, by the way, refers to Selenium RC.

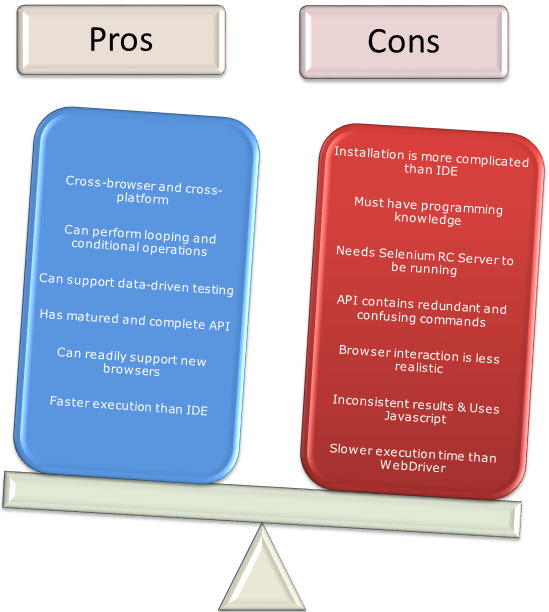
**Brief Introduction Selenium IDE**

Selenium Integrated Development Environment (IDE) is the **simplest framework** in the Selenium suite and is **the easiest one to learn**. It is a **Firefox plugin** that you can install as easily as you can with other plugins. However, because of its simplicity, Selenium IDE should only be used as a **prototyping tool**. If you want to create more advanced test cases, you will need to use either Selenium RC or WebDriver.



**Brief Introduction Selenium Remote Control (Selenium RC)**

Selenium RC was the **flagship testing framework** of the whole Selenium project for a long time. This is the first automated web testing tool that **allowed users to use a programming language they prefer**. As of version 2.25.0, RC can support the following programming languages:

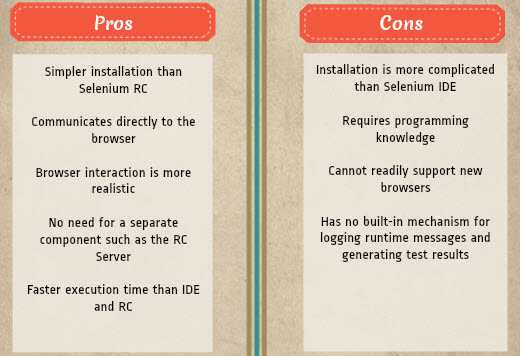
* Java,C#,PHP.Python,Perl,Ruby.
* 

**Brief Introduction WebDriver**

The WebDriver proves itself to be **better than both Selenium IDE and Selenium RC** in many aspects. It implements a more modern and stable approach in automating the browser's actions. WebDriver, unlike Selenium RC, does not rely on JavaScript for automation. **It controls the browser by directly communicating to it.**

The supported languages are the same as those in Selenium RC.

* Java
* C#
* PHP
* Python
* Perl
* Ruby



**Selenium Grid**

Selenium Grid is a tool **used together with Selenium RC to run parallel tests** across different machines and different browsers all at the same time. Parallel execution means running multiple tests at once.

**Features:**

* Enables **simultaneous running of tests** in **multiple browsers and environments.**
* **Saves time** enormously.
* Utilizes the **hub-and-nodes** concept. The hub acts as a central source of Selenium commands to each node connected to it.

**Waits:**

**Implicit Wait:  For the whole page**

**Explicit wait:  For a specific element**

**Fluent wait : for different intervals**

**WebDriver commands:**

* **Alert & Popup handling in Selenium:**

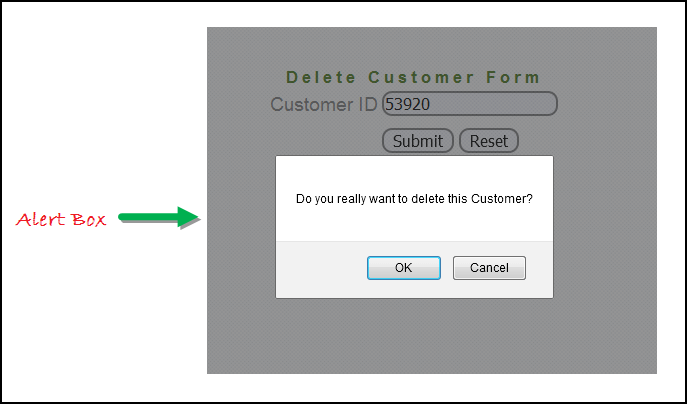
**What is Alert ?**

Alert is a small message box which displays on-screen notification to give the user some kind of information or ask for permission to perform certain kind of operation. It may be also used for warning purpose.

Here are few alert types:

**1) Simple Alert**

This simple alert displays some information or warning on the screen.



**2) Prompt Alert.**

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

**3) Confirmation Alert.**

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

**How to handle Alert in Selenium WebDriver**

Alert interface provides the below few methods which are widely used in Selenium Webdriver.

1) void dismiss() **// To click on the 'Cancel' button of the alert.**

driver.switchTo().alert().dismiss();

2) void accept() **// To click on the 'OK' button of the alert.**

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

3) String getText**() // To capture the alert message.**

driver.switchTo().alert().getText();

4) void sendKeys(String stringToSend) **// To send some data to alert box.**

driver.switchTo().alert().sendKeys("Text");

You can see a number of Alert methods are displayed as shown in below screen suggested by Eclipse.

We can easily switch to alert from the main window by using Selenium's **.switchTo()** method.

**Handling Keyboard & Mouse Events:**

**Handling special keyboard and mouse events are done using the Advanced User Interactions API**. It contains the **Actions** and the **Action** classes that are needed when executing these events. The following are the most commonly used keyboard and mouse events provided by the Actions class.

|  |  |
| --- | --- |
| **Method** | **Description** |
| **clickAndHold()** | Clicks (without releasing) at the current mouse location. |
| **contextClick()** | Performs a context-click at the current mouse location. |
| **doubleClick()** | Performs a double-click at the current mouse location. |
| **dragAndDrop(source, target)** | Performs click-and-hold at the location of the source element, moves to the location of the target element, then releases the mouse.  **Parameters:**  *source*- element to emulate button down at.  *target*- element to move to and release the mouse at. |
| **dragAndDropBy(source, x-offset, y-offset)** | Performs click-and-hold at the location of the source element, moves by a given offset, then releases the mouse.  **Parameters**:  *source*- element to emulate button down at.  *xOffset*- horizontal move offset.  *yOffset*- vertical move offset. |
| **keyDown(modifier\_key)** | Performs a modifier key press. Does not release the modifier key - subsequent interactions may assume it's kept pressed.  **Parameters**:  modifier*\_key* - any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| **keyUp(modifier \_key)** | Performs a key release.  **Parameters**:  modifier*\_key* - any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| **moveByOffset(x-offset, y-offset)** | Moves the mouse from its current position (or 0,0) by the given offset.  **Parameters**:  *x-offset*- horizontal offset. A negative value means moving the mouse left.  *y-offset*- vertical offset. A negative value means moving the mouse up. |
| **moveToElement(toElement)** | Moves the mouse to the middle of the element. **Parameters**:  *toElement*- element to move to. |
| **release()** | Releases the depressed left mouse button at the current mouse location |
|  |  |

**How to handle Selenium Popup window using Webdriver**

In automation, when we have multiple windows in any web application, the activity may need to switch control among several window from one to other in order to complete the operation. After completion of the operation, it has to return to the main window i.e. parent window. We will see this further in the article with an example.

In selenium web driver there are methods through which we can handle multiple windows.

**Driver.getWindowHandles();**

To handle all opened windows by web driver, we can use "Driver.getWindowHandles()" and then we can switch window from one window to another in a web application. Its return type is Iterator<String>.

**Driver.getWindowHandle();**

When the site opens, we need to handle the main window by **driver.getWindowHandle()**. This will handle the current window that uniquely identifies it within this driver instance. Its return type is String.

**Taking Screenshot using Webdriver**

**Its very important to take screenshot when we execute a test script. When we execute huge number of test scripts, and if some test fails, we need to check why the test has failed.**

**It helps us to debug and identify the problem by seeing the screen shot.**

**In selenium webdriver, we can take the screen shot using the below command.**

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

**FileUtils.copyFile(srcFile,new file(“path”);**

**Select api**

**Select api will allow you to choose from various options that are available among various drop downs all together.  It allows you to select all multiples are multiple as well.  Different options all together.**

* **First we find the drop down**
* **Create an object of select api by passing the element that you have caught and then use that api.**

**JavaScriptExecutor**

Some times some of the browsers you are automating they do not allow you to perform normal actions instead they ask you to do it by your javascript.

XPath, FireBug & FirePath

What is XPath:

**XPath is a language that describes a way to locate and process items in Extensible Markup Language (XML) documents by using an addressing syntax based on a path through the document’s logical structure or hierarchy. XPath in used in Selenium  to uniquely identify an element on a Webpage as element locator just like the way we use PostCode and House address in real world to locate Home Address.**

What is Firebug

**Firebug integrates with Firefox to put a wealth of web development tools at your fingertips while you browse. You can edit, debug, and monitor CSS, HTML, and JavaScript live in any web page. The whole content of this page is taken from the https://getfirebug.com/html.**

What is FirePath

**It is an extension to FireBug that adds a development tool to edit, inspect and generate XPath expressions and CSS3 Selectors.**

Why it is Useful to Selenium Automation Tester

**1) You can type self-written XPath and check if it is correct by highlighting the results directly on the Webpage.**

**2) Generate an XPath expression or a CSS selector for an element by right clicking on it and selecting “Inspect in FirePath” in the context menu.**

**3) Like Firebug it also gives you the Xpath of the selected Element.**

Difference between FireBug and FirePath

**The only difference from Automation tester point of view is FireBug returns Absolute XPath and FirePath returns Relative XPath.**

XPath Helper

**new tool called *XPath Helper* an *Add-On of Chrome Browser*. Even though the same task can be done by** [***Firefox***](http://toolsqa.wpengine.com/selenium-webdriver/how-to-add-extensions-firebug-firepath-to-firefoxdriver/) **and** [***Firepath***](http://toolsqa.wpengine.com/selenium-webdriver/xpath-firebug-firepath/) **plugin, but still in case you love to use Chrome Browser.**

**DropDown & Multiple Select Operations**

**It is just an ordinary operation like selecting any other type of element on a webpage.**

**You can choose it byID, Name, Css & Xpathetc. But to perform any action on this**

**Elements its required toimport**

**import org.openqa.selenium.support.ui.select package and to use it we need to create a new Select Object of class Select.**

**.**

**SELECTING DROPDOWN:**

**there are three ways to select the desired value in the dropdown amongst**

**the listed one.**

**selectByValue()**

**selectByVisibleText()**

**selectByIndex()**

**Select dropdown = new Select(driver.findElement(By.id("flowers")));**

**dropdown.selectByVisibleText("Aster");**

**or**

**dropdown.selectByIndex(1);**

**or**

**dropdown.selectByValue("Pink");**

**Benefits of Robot API**

**Robot API can simulate Keyboard and Mouse Event**

**Robot API can help in upload/download of files when using selenium web driver**

**Robot API can easily be integrated with current automation framework (keyword,**

**data-driven or hybrid)**

**AutoIT.**

**keypress(KeyEvent.VK\_Down)**

**mousePress(InputEvent.BUTTON3\_DOWN\_MASK) Right click**

**mouseRelease(InputEvent.BUTTON3\_DOWN\_MASK)**

**mousPress(InputEvent.BUTTON1\_DOWN\_MASK) left Click.**

**WHAT IS JAVASCRIPTEXECUTOR AND WHY DO WE USE IT:**

**JavaScriptExecutor is an interface which provides mechanism to execute**

**Javascript through selenium driver.**

**It provides “executescript” & "executeAsyncScript" methods, to run**

**JavaScript in the context of the currently selected frame or window**

**DIFFERENCE BETWEEN EXECUTESCRIPT AND EXECUTEASYNCSCRIPT**

**Execute an asynchronous piece of JavaScript in the context of the currently**

**selected frame or window.**

**Unlike executing synchronous JavaScript, scripts executed with this method**

**must explicitly signal they are finished by invoking the provided callback.**

**This callback is always injected into the executed function as the last**

**argument.**

**executeAsyncScript -> This method doesn't block the execution of next line**

**of code...till execution of this method is completed.**

**executeScript -> This method will block the execution till it's execution is**

**completed and then it moves to next line of code.**

**((JavascriptExecutor) driver).executeScript("alert('Hello');");**

**JavaSrcriptExecutor jse=(JavascriptExecutor)driver**

**jse.executeScript("window.scrollBy(0,-150);");**

**Alert Pop window**

**JavascriptExecutor js =(JavascriptExecutor)driver;**

**js.executeScript("alert('SW Test Academy!');");**

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

**Get Page Title**

**JavascriptExecutor js =(JavascriptExecutor)driver;**

**String title = js.executeScript("return document.title;").toString();**

**assertThat(driver.getTitle(), is(title));**

**Refresh Browser Window**

**JavascriptExecutor js =(JavascriptExecutor)driver;**

**js.executeScript("history.go(0);");**

generally two types of exception is there

checked exception-when prohrams knows that has chance to occur exception so need to use try catch exception –compiler time –able to handle

unchecked exception- run time exception- during exeution pf program ,will crash if it occurs-cant able to handle in that scenario idea to use try/catch when something can potentially go wrong

for example ther is file you are reading file then program will exeute second time ypur executing whicj overwrite all dat in file due to close file

1. If the exception was originated into the tested code:
   * If it was expected, declare it in the expected attribute of the Test annotation. Or, if further checks should be done on the exception object itself, catch it and ignore it. (In this case, there must be also a call to Assert.fail at the end of the try block, to indicate that the expected exception was not produced).
   * If it was not expected, catch it and execute Assert.fail. (A previous call to Exception.printStackTrace is also useful).
2. If the exception was not originated into the tested code or it is not interesting to the test (for example, most of the IOExceptions are produced at network level, before the test could *even be completed*), rethrow it at the throws clause.

There are 2 kinds of exceptions: Checked and Unchecked. A Checked exception **can be considered** one that is found by the compiler, and the compiler knows that it has a chance to occur, so you need to catch or throw it. For example, opening a file. It has a chance to fail, and the compiler knows this, so you're forced to catch or throw the possible IOException.

Dynamic elements

<http://www.testerlogic.com/handling-dynamic-elements-in-selenium-webdriver/>

production issue

<http://sqa.stackexchange.com/questions/16749/how-should-a-tester-deal-with-a-bug-found-in-production>

try to find out root cause analysis

Now find a solution to improve here and make it future proof.

Find a way of preventing this, and things like it from happening ever again.