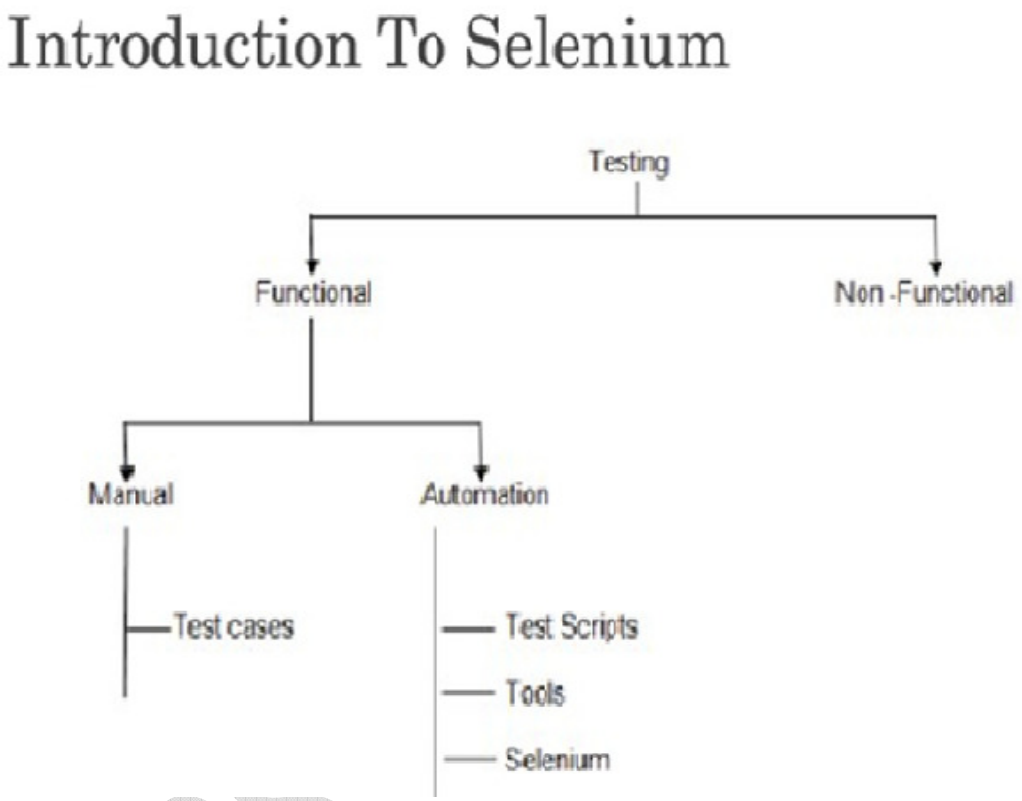
Selenium Hand Book

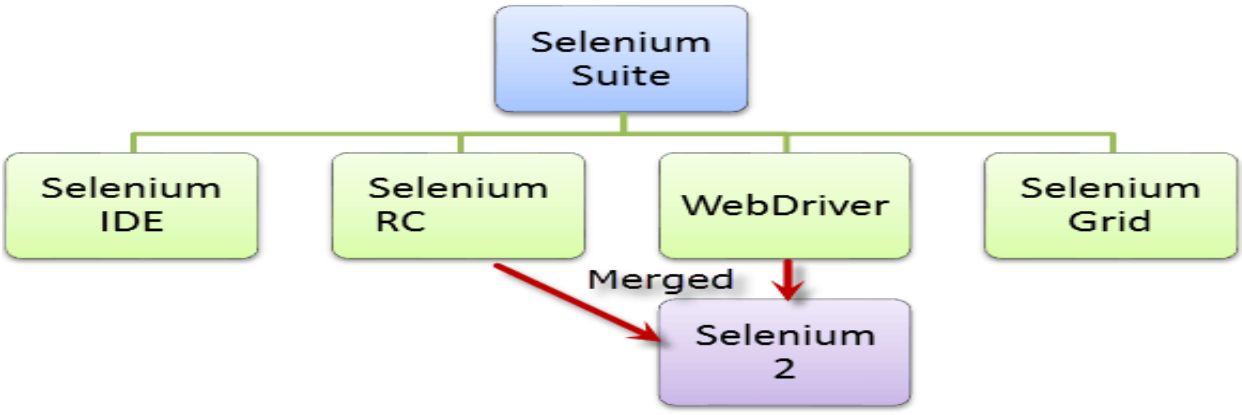
Contents

Introduction To 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 organisation. It has four components.



Selenium components

Introduction Selenium IDE

- Selenium Integrated Development Environment (IDE) is the simplest framework in the Selenium suite and isthe 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.

PROS	CONS
Very easy to use and install	Avilable onlly for Firefox
No Programming Expriance is required,knowledge of HTML and DOM is needed	Designed only to create Prototype of tests
Can export tests to formats usable in Selenium RC and webDriver	No support for Iteration and conditional operations
Has build in help and test results reporting module	Test execution is slow compare to selenium RC and WebDriver
Provide support for extensions	

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

PROS	CONS
Cross Browser and cross platform	Installation is more completed than IDE
Can perform looping and conditional operations	Must have programming knowledge
Can support data driven testing	Need Selenium RC server to be running
Has Matured and complite API	API Contains redudant and confusing commands
Can readly support new browsers	Browser interection is less realistic

Faster execution than IDE	Inconsistent result and uses javascript Slowe execution time than WebDriver

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

PROS	CONS
Simpler Installation than Selenium RC	Installation is more complicated than selinuim IDE
Communicates directly to the browser	Required programming knoowledge
Browser interection is more realistic	Cannot readly support new browser
No need to seprate component such as the RC server	Has no built in mechanism for logging runtime messages and generating test result
Faster exectuion time than IDE and RC	

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.

Browser and Environment Support

	Selenium IDE	**Selenium RC**	WebDriver
Browser Support	Mozilla Firefox	Mozilla Firefox Internet Explorer	Internet Explorer versions 6 to 9, both 32

		Google Chrome Safari Opera Konqueror Others	and 64-bit Firefox 3.0, 3.5, 3.6, 4.0, 5.0, 6, 7 and above (current version is 16.0.1) Google Chrome 12.0.712.0 and above (current version is 22.0.1229.94 m) Opera 11.5 and above (current version is 12.02) Android - 2.3 and above for phones and tablets (devices & emulators) iOS 3+ for phones (devices & emulators) and 3.2+ for tablets (devices & emulators) HtmlUnit 2.9 and above (current version is 2.10)
Operating System	Windows Mac OS X Linux	Windows Mac OS X Linux Solaris	All operating systems where the browsers above can run

Selenium Features

Open source: Works on multiple browsers and multiple operating systems as compared to other tools in market

Supports multiple platform and browser: You can develop selenium code and make it run parallely on multiple machines using differen browser.

Mobile testing : Support for Android and Iphone Testing

Simple: Selenium IDE is a simple tool which comes as an addon in firefox and is easy to use. It has the record and run feature which is very strong.

-You can also extend the functionality/scope of IDE with the help of many plugins available.

- You can also create your own Selenium IDE plugins

Webdriver is the latest version of selenium and is very strong. Its removed lots of drawbacks in RC and introduced many more features in selenium. - Selenium when used with Hudson can be used for Continuous integration
Object oriented datadriven or hybrid testing framework can be made very easily.
- You can use open source frameworks such as junit, testng, nuint etc and can write selenium test cases in them

QTP V/S SELENIUM

- **Quick Test Professional(QTP)** is a proprietary automated testing tool previously owned by the company **Mercury Interactive** before it was **acquired by Hewlett-Packard in 2006**. The Selenium Tool Suite has many advantages over QTP (as of version 11) as detailed below -

QTP	SELENIUM
Open source, free to use, and free of charge.	Commercial
Highly extensible	Limited add-ons
Can run tests across different browsers	Can only run tests in Firefox, Internet Explorer and Chrome
Supports various operating systems	Can only be used in Windows
Supports mobile devices	Supports mobile device using 3rd party software
Can execute tests while the browser is minimized	Needs to have the application under test to be visible on the desktop
Can execute tests in parallel.	Can only execute in parallel but using Quality Center which is again a paid product.

Advantages of QTP over Selenium

QTP	SELENIUM
Can test both web and desktop applications	Can only test web applications
Comes with a built-in object repository	Has no built-in object repository
Automates faster than Selenium because it is a fully featured IDE	Automates at a slower rate because it does not have a native IDE and only third party IDE can be used for development
Data-driven testing is easier to perform because it	Data-driven testing is more cumbersome since you

has built-in global and local data tables.	have to rely on the programming language's capabilities for setting values for your test data
Can access controls within the browser(such as the Favorites bar, Address bar, Back and Forward buttons, etc.)	Cannot access elements outside of the web application under test
Provides professional customer support	No official user support is being offered.
Has native capability to export test datainto external formats	Has no native capability to export runtime data onto external formats
Parameterization Support is in built	Parameterization can be done via programming but is difficult to implement.
Test Reports are generated automatically	No native support to generate test /bug reports

- Though clearly, QTP has more advanced capabilities, Selenium outweighs QTP in three main areas:
  - **Cost** (because Selenium is completely free)
  - **Flexibility** (because of a number of programming languages, browsers, and platforms it can support)
  - **Parallel testing** (something that QTP is capable of but only with use of Quality Center)

Scope and Growth of Selenium Testing

- Nowadays, software quality assurance (QA) has become an integral part of each project.
- Many organizations even adopt test-based development approach to make the coding compatible with testing modules.
- The QA professionals further use several test automation tools and frameworks to facilitate test management.
- As a compact and robust software testing framework, Selenium is used widely to web application.
- The open source framework was originally developed by a team of testers at ThoughtWorks in 2014.
- But within a decade, Selenium has become synonymous with internet application testing.
- A number of recent reports have highlighted that Selenium skills has become one of the most commonly and widely required test automation technology.
- As each enterprise has to optimize the look, feel and performance of a web application to deliver user experience, Selenium testing is expected to grow steadily over next few years.
- Why Selenium will Grow Steadily Over Next Few Years?
- Open Source and Free

- Selenium was released by ThoughtWork under the Apache 2.0 license. So the testing framework is open source, and can be downloaded by users without paying in charges.
- Support for Multiple Web Browsers and Operating Systems
- Each internet application must run across several web browsers and operating systems to keep the users engaged. So QA professionals look for a set of tools that allows them to quickly evaluate the performance of the application on some of the widely used web browsers.

A Complete IDE for Internet Application Testing : The Selenium IDE comes with a set of advanced tools that allows users to record, edit and debug tests without putting any extra effort. As the IDE was earlier known as Selenium Recorder

Client API for Several Programming Languages : The Selenium IDE records scripts in a special test scripting language called Selenese. But the testers have option to write the test scripts in a variety of programming languages in addition to Selenese.

The Selenium Client API supports a number of widely used programming languages like C#, Java, Perl, PHP, Ruby and Python

Comprehensive Documentation and Support : While migrating to a new web application testing framework, the QA professionals require additional training. The tools provided by Selenium are well documented. So the QA professionals can refer to a complete documentation and use guide to understand how the tools function.

Advantages of Selenium

- Open Source
- Supports all browsers like IE, Firefox, Mozilla, Safari
- Supports all Operating Systems.
- Supports all programming languages Java,Ruby,C# and Python.
- Run multiple test at a time

Testing framework

A testing framework or more specifically a testing automation framework is an execution environment for automated tests. It is the overall system in which the tests will be automated.

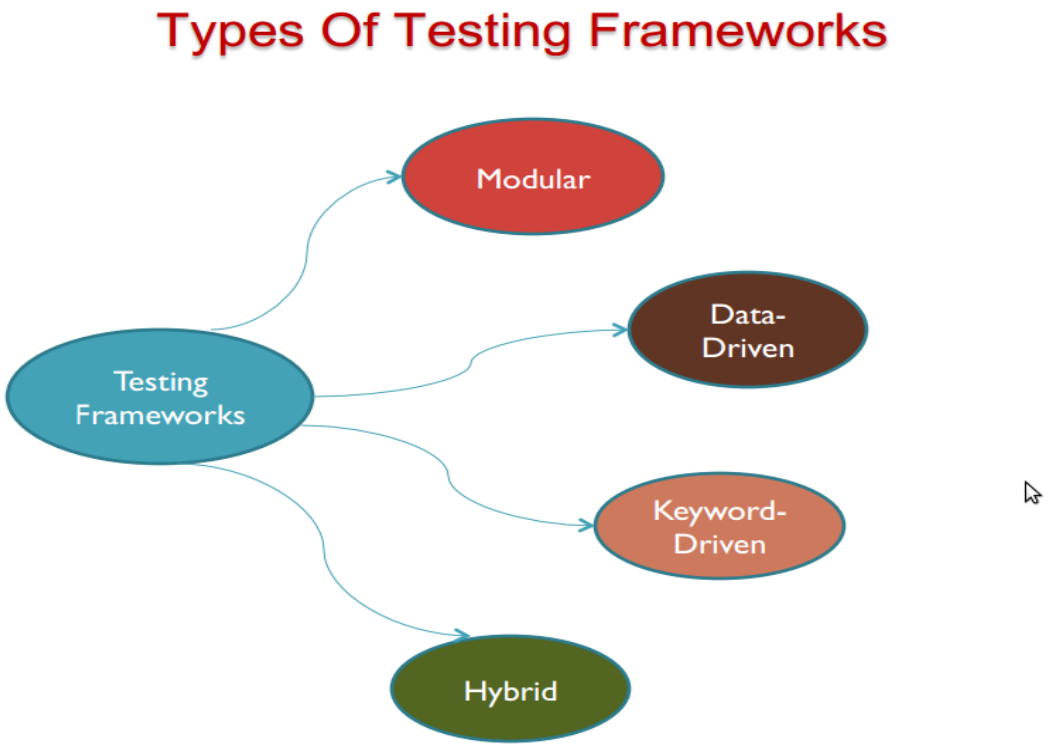
It is defined as the set of assumptions, concepts, and practices that constitute a work platform or support for automated testing.

The Testing framework is responsible for: Defining the format in which to express expectations. Creating a mechanism to hook into or drive the application under test Executing the tests Reporting results

- **Properties of a testing framework:**
  - It is application independent.
  - It is easy to expand, maintain and perpetuate/keep going.
- **Why the testing framework is required:**

- If we have a group of testers and suppose if each project implements a unique strategy then the time needed for the tester become productive in the new environment will take long.
- To handle this we cannot make changes to the automation environment for each new application that comes along.
- For this purpose we use a testing framework that is application independent and has the capability to expand with the requirements of each application.
- Also an organized test framework helps in avoiding duplication of test cases automated across the application.
- In short Test frameworks helps teams organize their test suites and in turn help improve the efficiency of testing.

- **Types Of Testing Frameworks:**



Modular Testing Framework

The Modularity testing framework is built on the concept of abstraction.

This involves the creation of independent scripts that represent the modules of the application under test. These modules in turn are used in a hierarchical fashion to build large test cases.

Thus it builds an abstraction layer for a component to hide that component from the rest of the application. Thus the changes made to the other part of the application do not effect that component.

- **Example of Modular Testing Framework:**

- To demonstrate the modular framework we use the calculator program.— Consider the basic functions of the calculator such as addition, subtraction, multiplication, division which are part of the Standard view.

We create scripts for these functions as follows:

Add:

```
Sub Main
Window Set Context, "Caption=Calculator", ""
PushButton Click, "ObjectIndex=10"  'Press 5
PushButton Click, "ObjectIndex=20"     'Press  +
PushButton Click, "ObjectIndex=14" 'Press  6
PushButton Click, "ObjectIndex=21" 'Press =
Result = LabelUP (CompareProperties, "Text=11.", "UP=Object Properties")
    'Compare Expected to Actual Results
End Sub
```

- **Advantages:**

Modular division of scripts leads to easier maintenance and also the scalability of the automated test suites. —

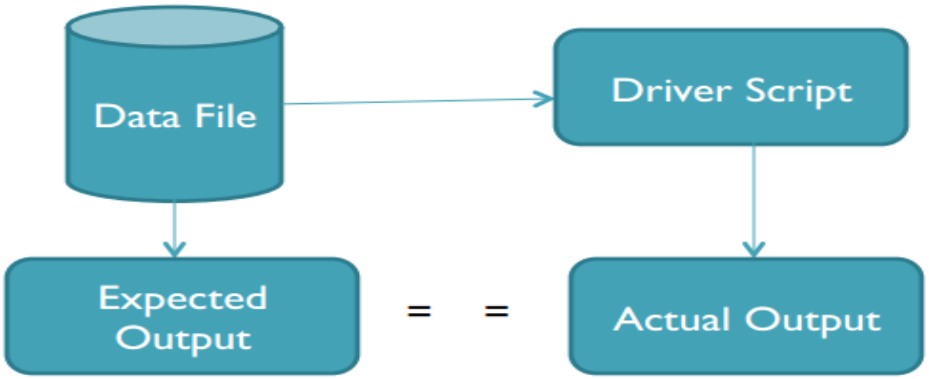
The functionality is available in easy to use test libraries so creating new driver scripts for different tests is easy and fast.

- **Disadvantages:**

The main problem with modular frameworks is that the test script have test data embedded in them. So when the test data needs to be updated we need to change the code of the script.

This becomes a big problem when the test script is large. For this purpose, data- driven testing frameworks have been introduced

Data-Driven Testing Framework



Data driven testing is where the test input and the expected output results are stored in a separate data file (normally in a tabular format) so that a single driver script can execute all the test cases with multiple sets of data.

The driver script contains navigation through the program, reading of the data files and logging of the test status information.

Example of Data Driven Testing Framework :

To demonstrate the data driven testing framework we use the login page of the flight reservation.
The first step involves creating the test data file. (testdata.csv). This data file contains the different types of input data which will be given to the driver script.

Test Case	Number1	Operator	Number2	Expected Result
Add	2	+	3	5
Subtract	3	-	2	1
Multiply	2	*	3	6
Divide	2	/	-2	-1

- In the next step we create a driver script and make references to the test data file.

```
data = open ( ' testdata.csv' ) . read ( )
l i n e s = data . s p l i t l i n e s ( )
#excluding the header row
        for line in lines:
            Read Number1
            Read Number2
            Read Operator
            Calculate the result using the Operator on
            Number 1 and Number2
```

- Compare the result to the expected result.

- This driver script reads the data from the data file computes the value and compares it with the expected result from the data file.
- **Advantages:**

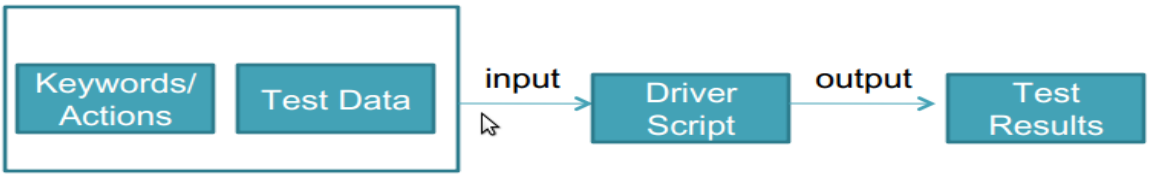
This framework reduces the number of overall test scripts needed to implement all the test cases.
Less amount of code is required to generate all the test cases.

- **Disadvantages:**

The test cases created are similar and creating new kind of tests requires creating new driver scripts that understand different data. Thus the test data and driver scripts are strongly related that changing either requires changing the other.
For this purpose keyword driven testing frameworks have been introduced.

- **Keyword- Driven Testing Framework:**

Keyword driven testing is an application independent framework utilizing data tables and self explanatory keywords to explain the actions to be performed on the application under test.
Not only is the test data kept in the file but even the directives telling what to do which is in the test scripts is put in external input data file.
These directives are called keywords. The keyword based testing is an extension to the data driven testing.



- **Example of Keyword Driven Testing Framework:**

To demonstrate the keyword driven testing we take the actions performed by the mouse when making calculations. We create a table that maps the actions performed with the mouse on the window of the calculator application. In this table-

1.The windows column represents the application for which we are performing the mouse action.
2.The control column represents the control that we are clicking with the mouse.
3.The action column represents the action performed by the mouse.
4.The argument column contains the specific control value

Window	Control	Action	Arguments
Calculator	Menu		View, Standard
Calculator	Pushbutton	Click	2
Calculator	Pushbutton	Click	+
Calculator	Pushbutton	Click	3
Calculator	Pushbutton	Click	=
Calculator		Verify Result	5
Calculator		Clear	
Calculator	Pushbutton	Click	5
Calculator	Pushbutton	Click	-
Calculator	Pushbutton	Click	3
Calculator	Pushbutton	Click	=
Calculator		Verify Result	2

After creating the table, we create a set of scripts for reading in the table, executing each step based on the keyword contained in the action field and logs any relevant information.
The below pseudo code represents this test of scripts.

- Main Script / Program

- Connect to data tables.

Read in row and parse out values.

- Pass values to appropriate functions.
- Close connection to data tables.

Advantages:

Automation expertise is not required to maintain or create a new set of test cases.

- Keywords are reused across multiple test cases.

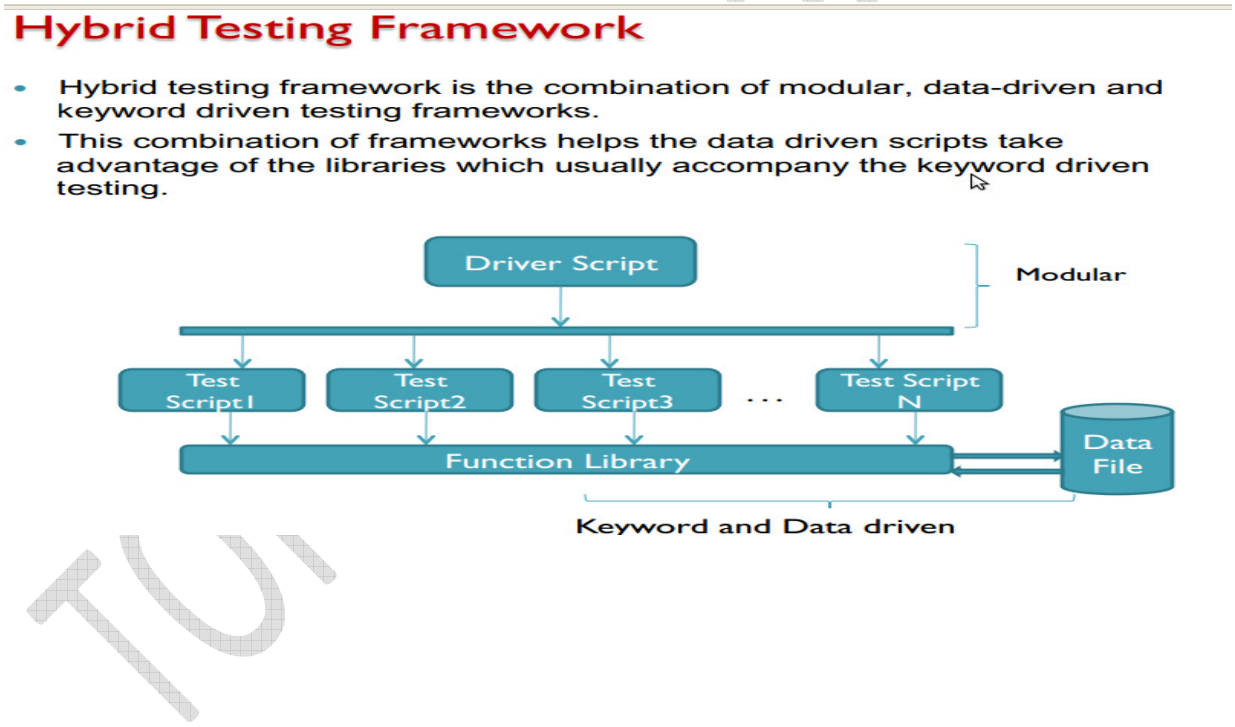
Disadvantages:

The main problem is that this requires a more complicated framework than the data driven framework.

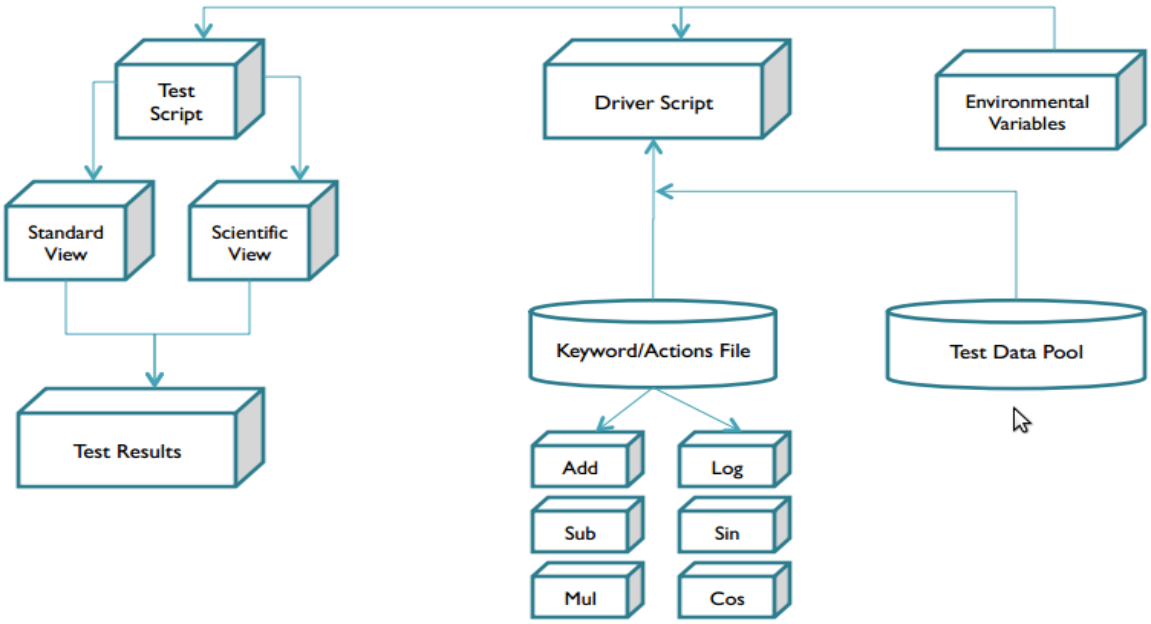
- With the keyword driven approach the test cases get longer and complex and this is due to the greater flexibility that this approach offers.

Hybrid Testing Framework

Hybrid testing framework is the combination of modular, data-driven and keyword driven testing frameworks. This combination of frameworks helps the data driven scripts take advantage of the libraries which usually accompany the keyword driven testing.



The hybrid framework for the calculator can be shown as follows:



Comparison of Frameworks

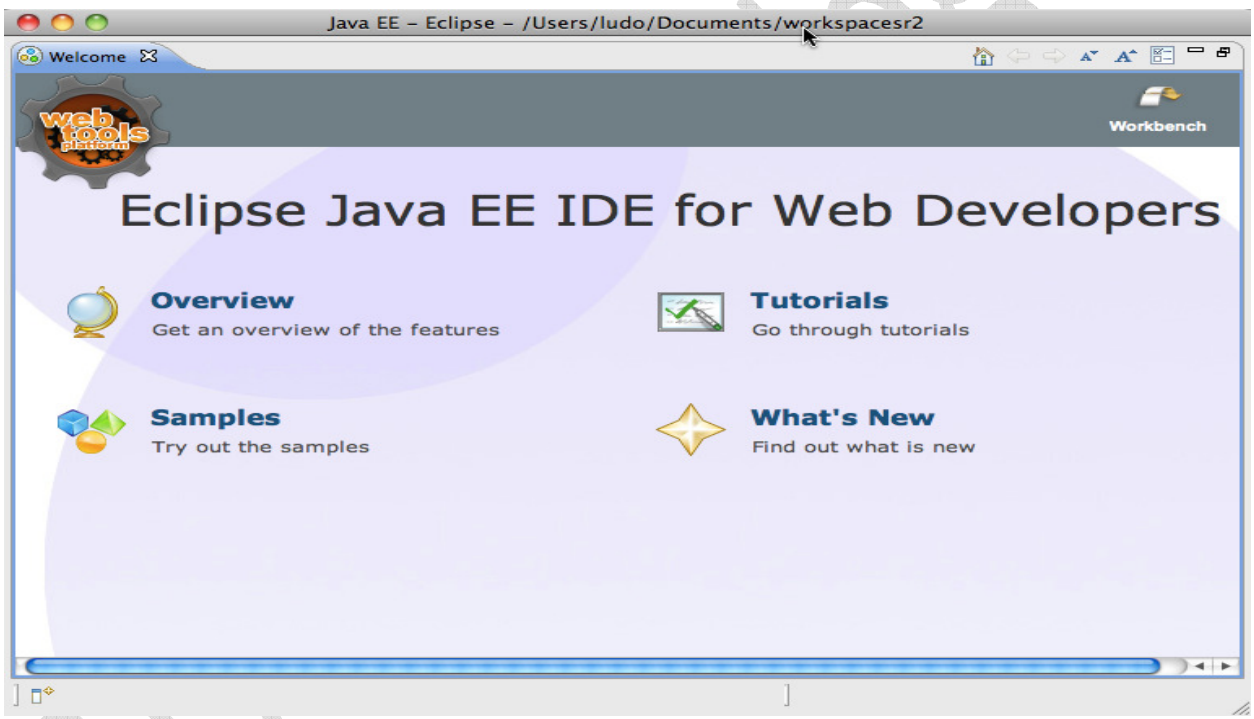
Approach	Advantages	Disadvantages
Modular testing framework	Modular approach Reusable functions Hierarchical Structure	Test data within the scripts limits reusability, Test script is dependent on software.
Data driven testing framework	Improved Maintainability	Dependency on technical expertise, Test script is dependent on software.
Keyword driven testing framework	Ease of maintenance, Scalability, Less dependency of software.	Dependency on technical expertise, Requires large effort
Hybrid testing framework	Integrates the advantages of all the other frameworks.	Increased Complexity

Overview of Eclipse

In the context of computing, Eclipse is an integrated development environment (IDE) for developing applications using the Java programming language and other programming languages such as C/C++, Python, PERL, Ruby etc.

The Eclipse platform which provides the foundation for the Eclipse IDE is composed of plug-ins and is designed to be extensible using additional plug-ins. Developed using Java, the Eclipse platform can be used to develop rich client applications, integrated development environments and other tools. Eclipse can be used as an IDE for any programming language for which a plug-in is available.

Eclipse platform and other plug-ins from the Eclipse foundation is released under the Eclipse Public License (EPL). EPL ensures that Eclipse is free to download and install. It also allows Eclipse to be modified and distributed.



Selenium RC Architecture

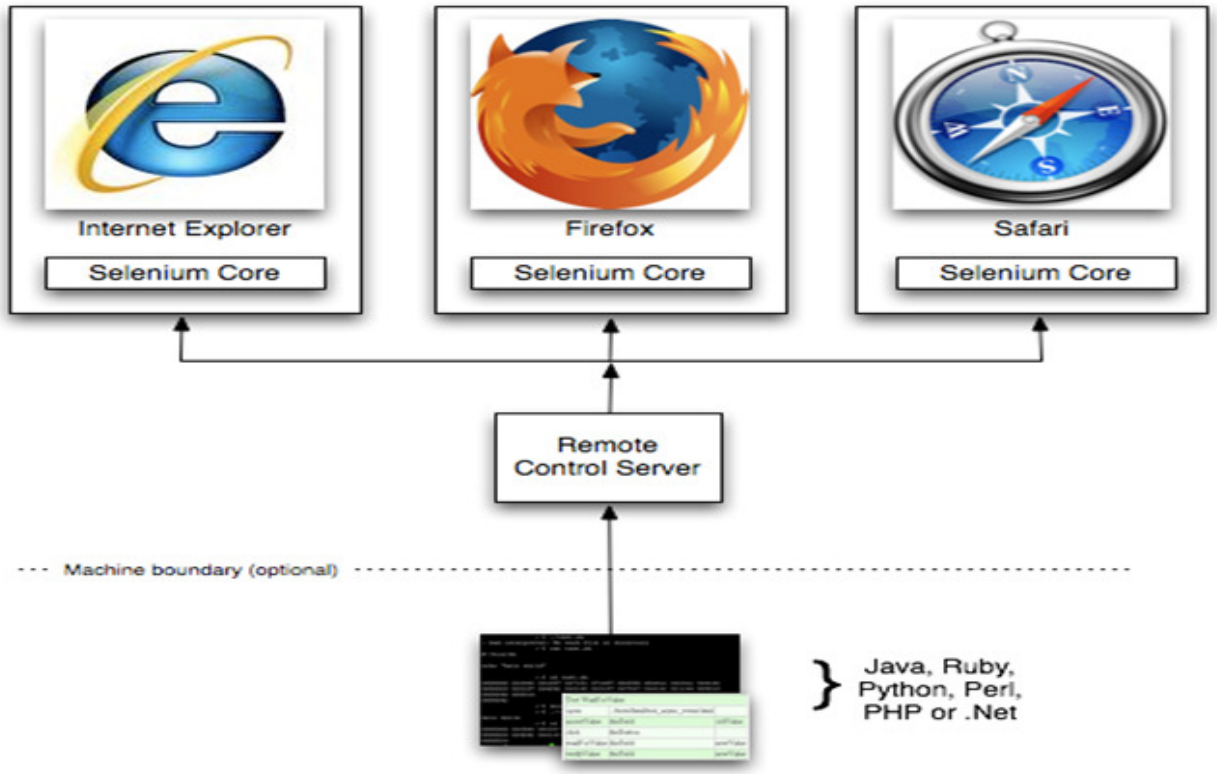
Selenium Remote Control is a test tool that allows you to write automated web application UI tests in any programming language against any HTTP website using any mainstream JavaScript-enabled browser.

Selenium RC comes in two parts:

- A server which automatically launches and kills browsers, and acts as a HTTP proxy for web requests from them.

Client libraries for your favourite computer language

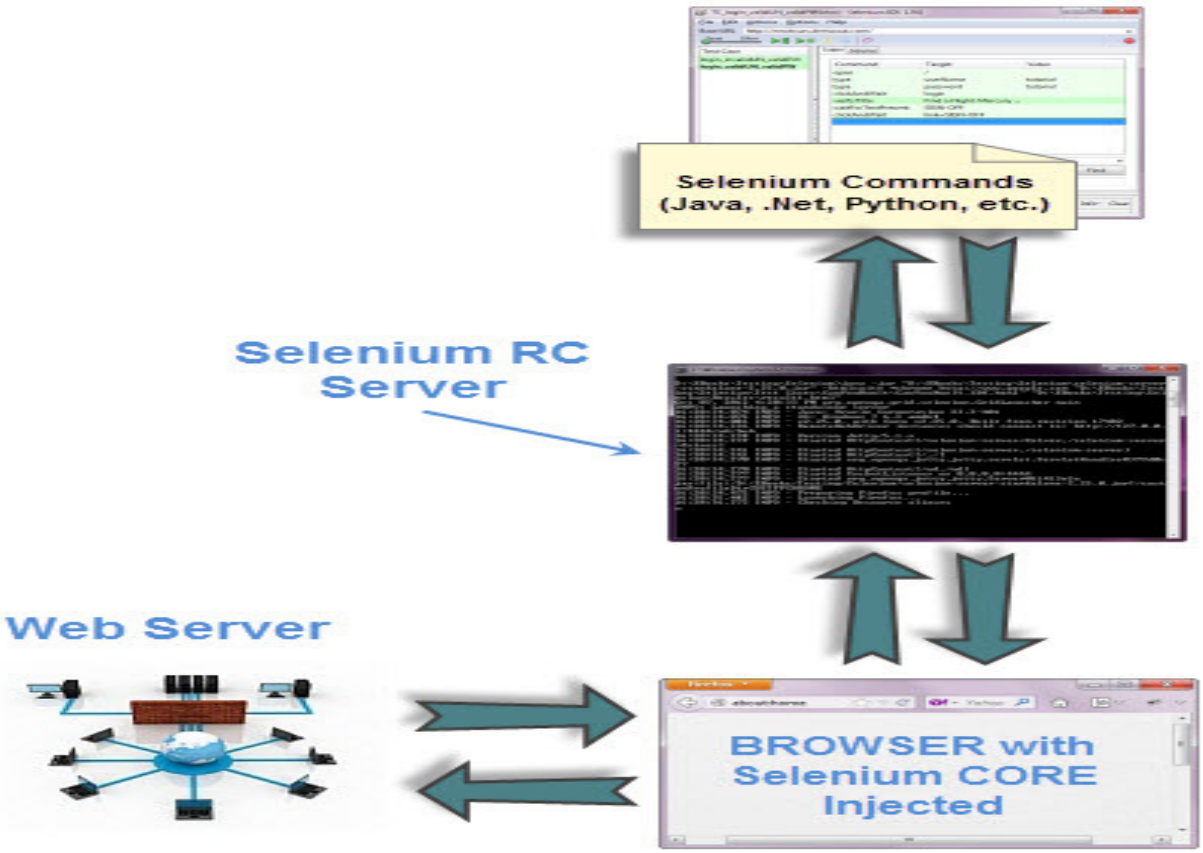
Below is a simplified architectural representation.



Selenium RC's architecture is way more complicated:

- You first need to launch a separate application called Selenium Remote Control (RC) Server before you can start testing
- The Selenium RC Server acts as a "middleman" between your Selenium commands and your browser
- When you begin testing, Selenium RC Server "injects" a Javascript program called Selenium Core into the browser.
- Once injected, Selenium Core will start receiving instructions relayed by the RC Server from your test program.
- When the instructions are received, Selenium Core will execute them as Javascript commands.

- The browser will obey the instructions of Selenium Core, and will relay its response to the RC Server.
- The RC Server will receive the response of the browser and then display the results to you.
- RC Server will fetch the next instruction from your test script to repeat the whole cycle.



- Brief explanation about advantages of web driver
    - Selenium uses JavaScript to automate web pages. This lets it interact very tightly with web content, and was one of the first automation tools to support Ajax and other heavily dynamic pages.
    - However, this also means Selenium runs inside the JavaScript sandbox. This means you need to run the Selenium-RC server to get around the same-origin policy, which can sometimes cause issues with browser setup.
- WebDriver on the other hand uses native automation from each language. While this means it takes longer to support new browsers/languages, it does offer a much closer 'feel' to the browser.
- If you're happy with WebDriver, stick with it, it's the future. There are limitations and bugs right now, but if they're not stopping you, go for it.

Primitive data types are predefined by the language and named by a keyword.

Datatypes	Discription
byte	8-bit signed two's complement integer. Range (-128 (-2^7) to 127 (inclusive)(2^7 -1)). Default value is 0
Short	16-bit signed two's complement integer. Range -32,768 (-2^15) to 32,767 (inclusive) (2^15 -1). Default value is 0.
Int	32-bit signed two's complement integer. Range-2,147,483,648.(2^31)to2,147,483,647(inclusive).(2^31 -1).Default value is 0.
Long	64-bit signed two's complement integer. Range -9,223,372,036,854,775,808.(-2^63) to 9,223,372,036,854,775,807 (inclusive). (2^63 -1). Default value is 0L.
Float	single-precision 32-bit IEEE 754 floating point. Default value is 0.0f.
Double	double-precision 64-bit IEEE 754 floating point. Default value is 0.0d.
Boolean	boolean data type represents one bit of information either true or false. Default value is false.
char	single 16-bit Unicode character. Range '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).

- In Java a reference data type is a variable that can contain the reference or an address of dynamically created object. The reference data types are arrays, classes and interfaces.
- Class type

e.g.

```
public class Person
{ .....
Person p=new Person();//p is of class type variable.
}
```

- Array Type
  - An array is a special kind of object that contains values called elements.

- The java array enables the user to store the values of the same type in contiguous memory allocations.

e.g.
int [] a = new int [10];
String [] cities = {—Ahmedabad",—Mumbai", —Pune"};

- **Modifiers**
  - **public,**
  - **private,**
  - **Protected,**
  - **Default**

Modifier	**Same Class**	**Same Package**	**Within Subclass outside the package**	**World**
public	**y**	**y**	**y**	**y**
Protected	**y**	**y**	**y**	**n**
no modifier	**y**	**y**	**n**	**n**
private	**y**	**n**	**n**	**n**

Assignments and Operators

- The Simple Assignment Operator
- One of the most common operators that you'll encounter is the simple assignment operator "=". You saw this operator in the Bicycle class; it assigns the value on its right to the operand on its left:

int cadence = 0;
int speed = 0;
int gear = 1;

- This operator can also be used on objects to assign *object references.*
- *Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:*
  - Arithmetic Operators
  - Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

The Arithmetic Operators:

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators:

Assume integer variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	A + B will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10
*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0
/	Division - Divides left hand operand by right hand operand	B / A will give 2
++	Increment - Increases the value of operand by 1	B++ gives 21
--	Decrement - Decreases the value of operand by 1	B-- gives 19

- **The Relational Operators:** There are following relational operators supported by Java language
  - Assume variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true	A==b is not true
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true	A!=b is true
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	a>b is not true
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	A<b is true
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true	a>=b is not true
<=	Checks if the value of left	A<=b is true

	operand is less than or equal to the value of right operand, if yes then condition becomes true.	

- **The Bitwise Operators:**
  - Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.
  - Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60; and b = 13; now in binary format they will be as follows:

a = 0011 1100
b = 0000 1101
---
a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a = 1100 0011

- **The Logical Operators:**
  - The following table lists the logical operators:
  - Assume Boolean variables A holds true and variable B holds false, then:

Operator	Description	example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A&&b) is false
||	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A||b) is true
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false	!(a&&b) is true

- **The Assignment Operators:**
  - There are following assignment operators supported by Java language:

Operator	Description	example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	operand to the left operand and	C += A is equivalent to C = C + A

	assign the result to left operand	
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand .	C -= A is equivalent to C = C - A
*=	right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A.
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator .	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2

- **Misc Operators:**
  - There are few other operators supported by Java Language.

Conditional Operator

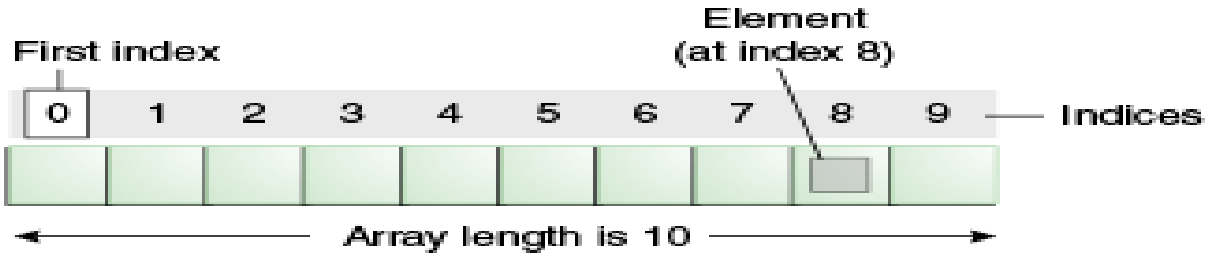
Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide which value should be assigned to the variable. The operator is written as:

variable x = (expression) ? value if true : value if false

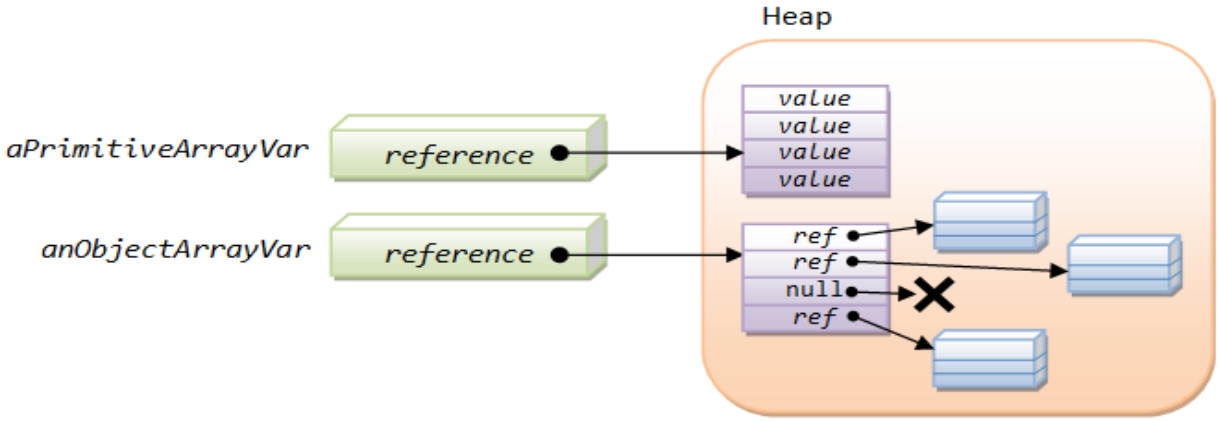
Arrays:

- **Introduction:**
  - An array is a container object that holds a fixed number of values of a single type.
  - The length of an array is established when the array is

- created.
- After creation, its length is fixed.



- **Advantages of Array:**
- Arrays are most appropriate for storing a fixed amount of data which will be accessed in an unpredictable fashion.
- Arrays that has become very important on modern architectures is that iterating through an array has good locality of reference.
- Arrays are much faster than iterating through a linked list of the same size, which tends to jump around in memory.
- Arrays also are among the most compact data structures; storing 100 integers in an array takes only 100 times the space required to store an integer



- **Array Types**

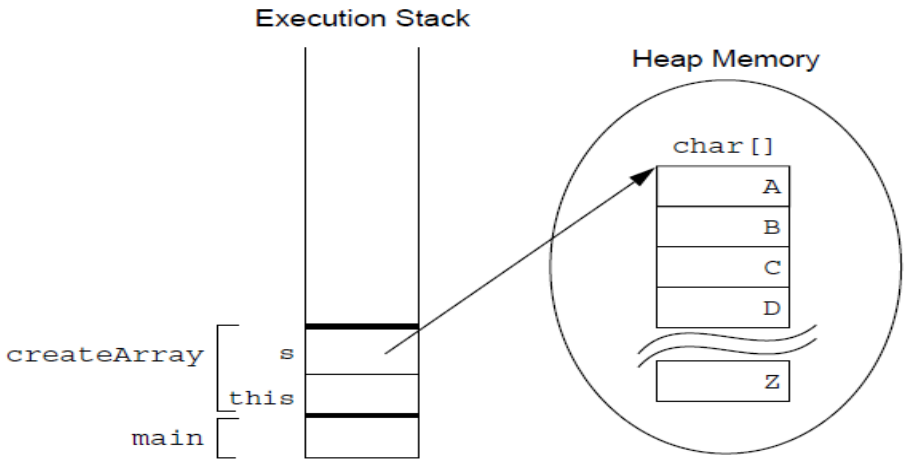
  There are 2 types of array available in java.
  - Primitive type,
    - 1. byte b; // byte is a primitive type
    - 2. byte[] arrayOfBytes; // byte[] is an array type: array of byte
    - 3. byte[][] arrayOfArrayOfBytes; // byte[][] is another type: array of byte[]
  - Class/Object type.
    - Point[] points;
- Primitive Types and Reference type Arrays.
  - char s[];
  - Point p[];
  - char[] s;

- Point[] p;

- Primitive Array

```
public char[] createArray() {
char[] s;
s = new char[26];
for ( int i=0; i<26; i++ ) {
s[i] = (char) ('A' + i);
}
return s;
}
```

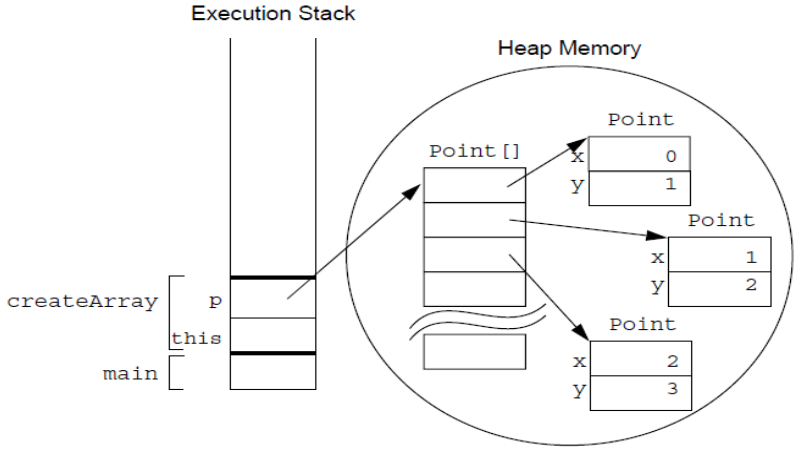
Creating an Array of Character Primitives



- **Reference type array:**

```
public Point[] createArray() {
Point[] p;
p = new Point[10];
for ( int i=0; i<10; i++ ) {
p[i] = new Point(i, i+1);
}
return p;
}
```

Creating an Array of Character Primitives With `Point` Objects



- **Creating Reference Arrays**

Another example, an object array:

```
public Point[] createArray() {
Point[] p;
p = new Point[10];
for ( int i=0; i<10; i++ ) {
p[i] = new Point(i, i+1);
}
return p;
}
```

- The String class represents character strings.
  - Strings are constant; their values cannot be changed after they are created.
  - Some important methods.

It's immutable/fixed class

Funcation	**Description**
char charAt(int index)	Returns the char value at the specified index. An index ranges from 0 to length() - 1
public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)	Copies characters from this string into the destination character array.
public void getBytes(int srcBegin, int srcEnd, byte[] dst,	Copies characters from this string into the destination byte array.

int dstBegin)	
public boolean equalsIgnoreCase(String anotherString)	Compares this String to another String, ignoring case considerations.
public String substring(int beginIndex)	Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string
public String concat(String str)	Concatenates the specified string to the end of this string.
public String replace(char oldChar, char newChar)	Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar
split public String[] split(String regex, int limit)	Splits this string around matches of the given regular expression.
public String toLowerCase()	Converts all of the characters in this String to lower case.
public String toUpperCase()	Converts all of the characters in this String to upper case
public String trim()	Returns a copy of the string, with leading and trailing whitespace omitted.
public char[] toCharArray()	Converts this string to a new character array.
public static String valueOf(char c)	Returns the string representation of the char argument.

Java Fucntions/Methods

A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.println method, for example, the system actually executes several statements in order to display a message on the console.

Now you will learn how to create your own methods with or without return values, invoke a method with or without parameters, overload methods using the same names, and apply method abstraction in the program design

Creating Method:

- Considering the following example to explain the syntax of a method:
    - public static int funcName(int a, int b)

    {

        // body

    }
- Here,
    - public static : modifier.
    - int: return type
    - funcName: function name
    - a, b: formal parameters
    - int a, int b: list of parameters
- Methods are also known as Procedures or Functions:
    - Procedures: They don't return any value.
    - Functions: They return value.
- Method definition consists of a method header and a method body. The same is shown below:

```
modifier returnType nameOfMethod (Parameter List)
 {
 // method body
}
```

- The syntax shown above includes:

modifier: It defines the access type of the method and it is optional to use.
returnType: Method may return a value.
nameOfMethod: This is the method name. The method signature consists of the method name and the parameter list.
Parameter List: The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.
method body: The method body defines what the method does with statements.

- **Example:**

Here is the source code of the above defined method called max(). This method takes two parameters num1 and num2 and returns the maximum between the two:

```
    /** the snippet returns the minimum between two numbers */
public static int minFunction(int n1, int n2)
{
int min;
if (n1 > n2)
    min = n2;
else
    min = n1;
 return min;
```

}

- **Method Calling:**

For using a method, it should be called. There are two ways in which a method is called i.e. method returns a value or returning nothing (no return value).

- The process of method calling is simple. This called method then returns control to the caller in two conditions, when:
  - return statement is executed.
  - reaches the method ending closing brace.
- The methods returning void is considered as call to a statement. Lets consider an example:
  - System.out.println("This is tutorialspoint.com!");
- The method returning value can be understood by the following example:
  - int result = sum(6, 9);

Example:

Following is the example to demonstrate how to define a method and how to call it:

```
public class ExampleMinNumber{
  public static void main(String[] args) {
    int a = 11;
    int b = 6;
    int c = minFunction(a, b);
    System.out.println("Minimum Value = " + c);
  }
 /** returns the minimum of two numbers */
 public static int minFunction(int n1, int n2) {
    int min;
    if (n1 > n2)
      min = n2;
    else
      min = n1;
     return min;
  }
}
```

This would produce the following result:

```
Minimum value = 6
```

- **The void Keyword:**

The void keyword allows us to create methods which do not return a value. Here, in the following example we're considering a void method *methodRankPoints*. This method is a void method which does not return any value. Call to a void method must be a statement i.e.*methodRankPoints(255.7);*. It is a Java statement which ends with a semicolon as shown below.

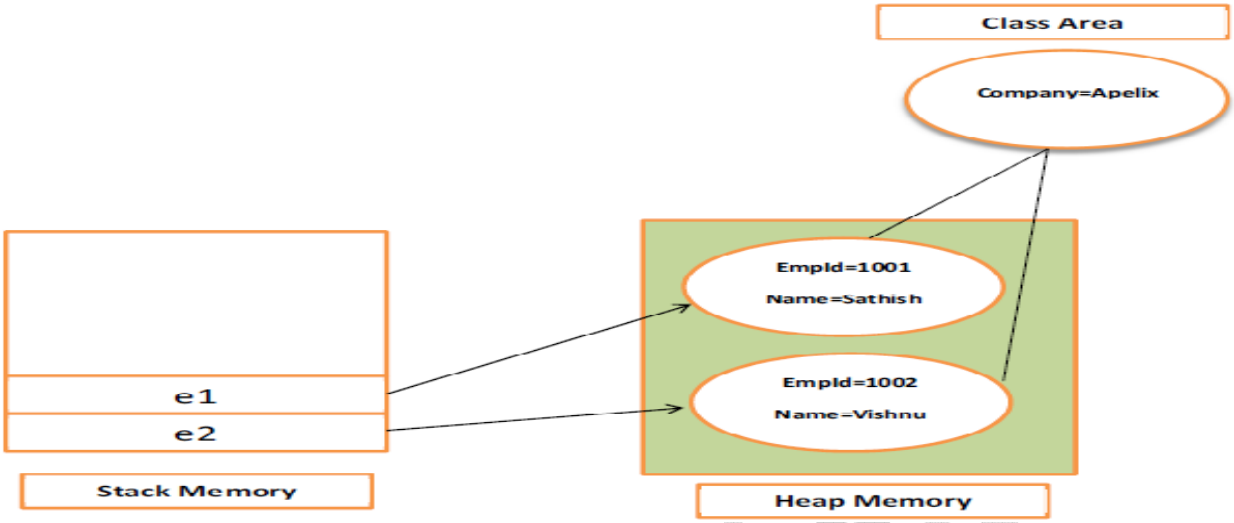
- **Example:**

```
public class ExampleVoid
```

```
{
        public static void main(String[] args)
        {
                methodRankPoints(255.7);
        }

        public static void methodRankPoints(double points)
        {
                if (points >= 202.5)
                {
                 System.out.println("Rank:A1");
                 }
                 else if (points >= 122.4)
                {
                 System.out.println("Rank:A2");
                }
                else
                {
                 System.out.println("Rank:A3");
                 }
         }
 }
```

Static,Final

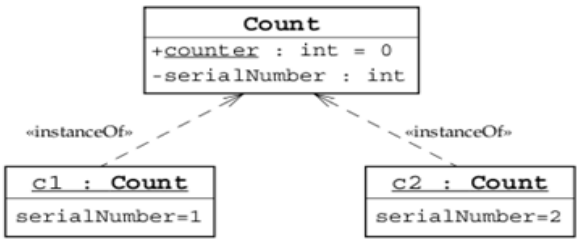
- "static" keyword.
- The static keyword is used as a modifier on
    - variables,
    - methods,
    - nested classes & blocks.
    - The static keyword declares the attribute or method isassociated with the class as a whole rather than any particular instance of that class.
    - Thus static members are often called class members, such as class attributes or class methods.



- **"static" variable**

- Java instance variables are given separate memory for storage.
- If there is a need for a variable to be common to all the objects of a single java class, then the static modifier should be used in the variable declaration.
- Any java object that belongs to that class can modify its static variables.
- Also, an instance is not a must to modify the static variable and it can be accessed using the java class directly.
- Static variables can be accessed by java instance methods also.
- When the value of a constant is known at compile time it is declared_final' using the_static' keyword.

Class attributes are shared among all instances of a class:




```
public class Count {
  private int serialNumber;
  public static int counter = 0;

  public Count() {
    counter++;
    serialNumber = counter;
  }
}
```

Static Methods

- Similar to static variables, java static methods are also common to classes and not tied to a java instance.
- Good practice in java is that, static methods should be invoked with using the class name though it can be invoked using an object.
- ClassName.methodName(arguments) or
- objectName.methodName(arguments)
- General use for java static methods is to access static fields.
- Static methods can be accessed by java instance methods.
- Java static methods cannot access instance variables or instance methods directly.
- Java static methods cannot use the‗this' keyword.

```
public class Counter  {

private static int counter = 0;
public static int getTotalCount(){
      return counter;
 }

public Counter(){
 counter++;
 }  }
 public class Tester {
 public static void main(String[] args) {
 System.out.println("total counter"+Counter.getTotalCount());
 }
}
```

Java Static Classes For java classes, only an inner class can be declared using the static modifier. For java a static inner class it does not mean that, all their members are static.
These are called nested static classes in java.

```
class OuterClass
{
private static String msg = "Hello";
public static class NestedStaticClass
{
public void printMessage()
{
System.out.println("Message from nested static class: " + msg);
    }
}
public class InnerClass {

public void display() {
System.out.println("Message from non-static nested class: " + msg);
}
}
}
 class Main {

public static void main(String args[]) {
 OuterClass.NestedStaticClass printer = new
 OuterClass.NestedStaticClass();
printer.printMessage();
 OuterClass outer = new OuterClass();
 OuterClass.InnerClass inner = outer.new InnerClass();
 inner.display();
 } }
```

"**static" initialize block**

- A class can contain code in a static block that does not exist within a method body.
- Static block code executes once only, when the class isbloaded.
- Usually, a static block is used to initialize static (class) attributes.

```
public class Test {

    static
    {
    System.out.println("Static");
    }

    {
    System.out.println("Non-static block");
```

```
        }
        public static void main(String[] args)
        {
        Test t = new Test();
        Test t2 = new Test();
        }
        }
```

- **Final**
- final‘ modifier can be used with
    - Class,
    - Variable &
    - Method.

```
 final class
        This class can not extended.
        public final class MyFinalClass
        {...
}
// forbidden
        public class subclass extends MyFinalClass
{...
}
```

- **final variable**

A final variable is a constant.

A variable that is declared as final and not initialized is called a blank final variable.

A blank final variable forces the constructors to initialise it.

Example

```
public class Bank
{
private static final double DEFAULT_INTEREST_RATE = 3.2;
// more declarations
}
```

final method

- You cannot override a final method.
- A blank final method variable must be set in the method body before being used.

```
public class MyClass  {
public void myMethod() {...}
public final void
myFinalMethod() {...}  }
public class AnotherClass extends MyClass {

public void myMethod() {...}
 // Ok
 public final void myFinalMethod() {...} // forbidden }
```

- **OOPS**

Object Oriented Programming System

Object means a real word entity such as pen, chair, table etc.

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects.

It simplifies the software development and maintenance by providing some concepts:

OOPS concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Name	Description
Object	Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.
Class	**Collection of objects** is called class. It is a logical entity.
Inheritance	**When one object acquires all the properties and behaviours of parent object** i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.
Polymorphism	When **one task is performed by different ways** i.e. known as polymorphism. For example: to convense the customer differently, to draw something e.g. shape or rectangle etc. In java, we use method overloading and method overriding to achieve polymorphism. Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.
Abstraction	**Hiding internal details and showing functionality** is known as abstraction. For example: phone call, we don't know the internal processing. In java, we use abstract class and interface to achieve abstraction.
Encapsulation	Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different

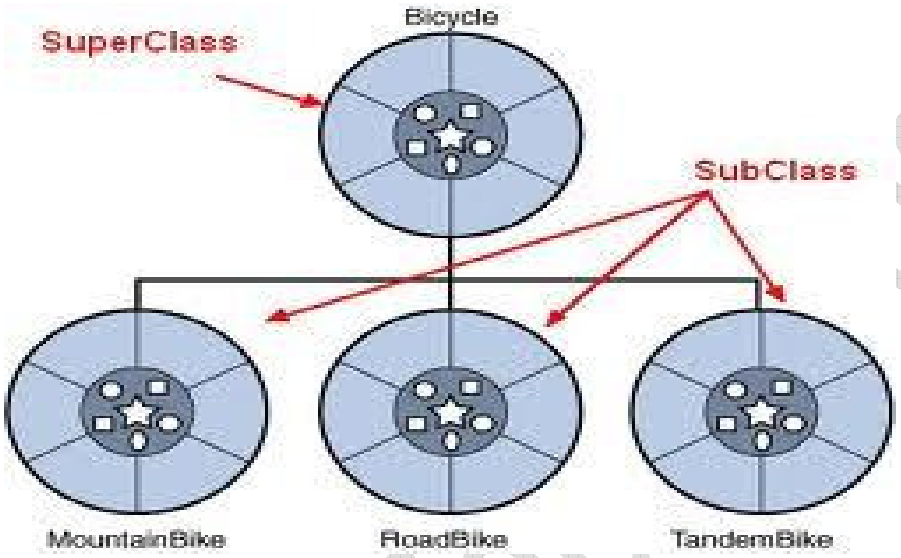
	medicines. A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here

Inheritance

Inheritance is a mechanism wherein a new class is derived from an existing class.

In Java, classes may inherit or acquire the properties and methods of other classes.

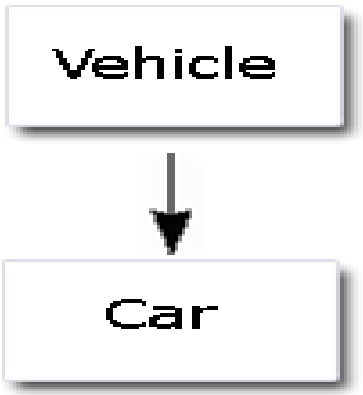
A class derived from another class is called a subclass,whereas the class from which a subclass is derived is called a super class.



Reusability	facility to use public methods of base class without rewriting the same.
Extensibility	extending the base class logic as per business logic of the derived class.
Data hiding	base class can decide to keep some data private so that it cannot be altered by the derived class.
Overriding	With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class

The following kinds of inheritance are there in java.

Simple Inheritance



- When a subclass is derived simply from it's parent class then this mechanism is known as simple inheritance.
- In case of simple inheritance there is only a sub class and it's parent class.

It is also called single inheritance or one level inheritance

Multilevel Inheritance

It is the enhancement of the concept of inheritance. When a subclass is derived from a derived class then this mechanism is known as the multilevel inheritance.
The derived class is called the subclass or child class for it‘s parent class and this parent class works as the child class for it's just above (parent) class.
Multilevel inheritance can go up to any number of level.

Java does not support multiple Inheritance.
Multiple Inheritance
The mechanism of inheriting the features of more than one base class into a single class is known as multiple inheritance.
Java does not support multiple inheritance but the multiple inheritance can be achieved by using the interface.
In Java Multiple Inheritance can be achieved through use of Interfaces by implementing more than one interfaces in a class.
Interface:

- An interface is a collection of abstract methods.
- A class implements an interface, thereby inheriting the abstract methods of the interface.
- Class describes the attributes and behaviors of an object.
- An interface contains behaviors that a class implements.

Class vs. Interface

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface is not extended by a class; it is implemented by a class.

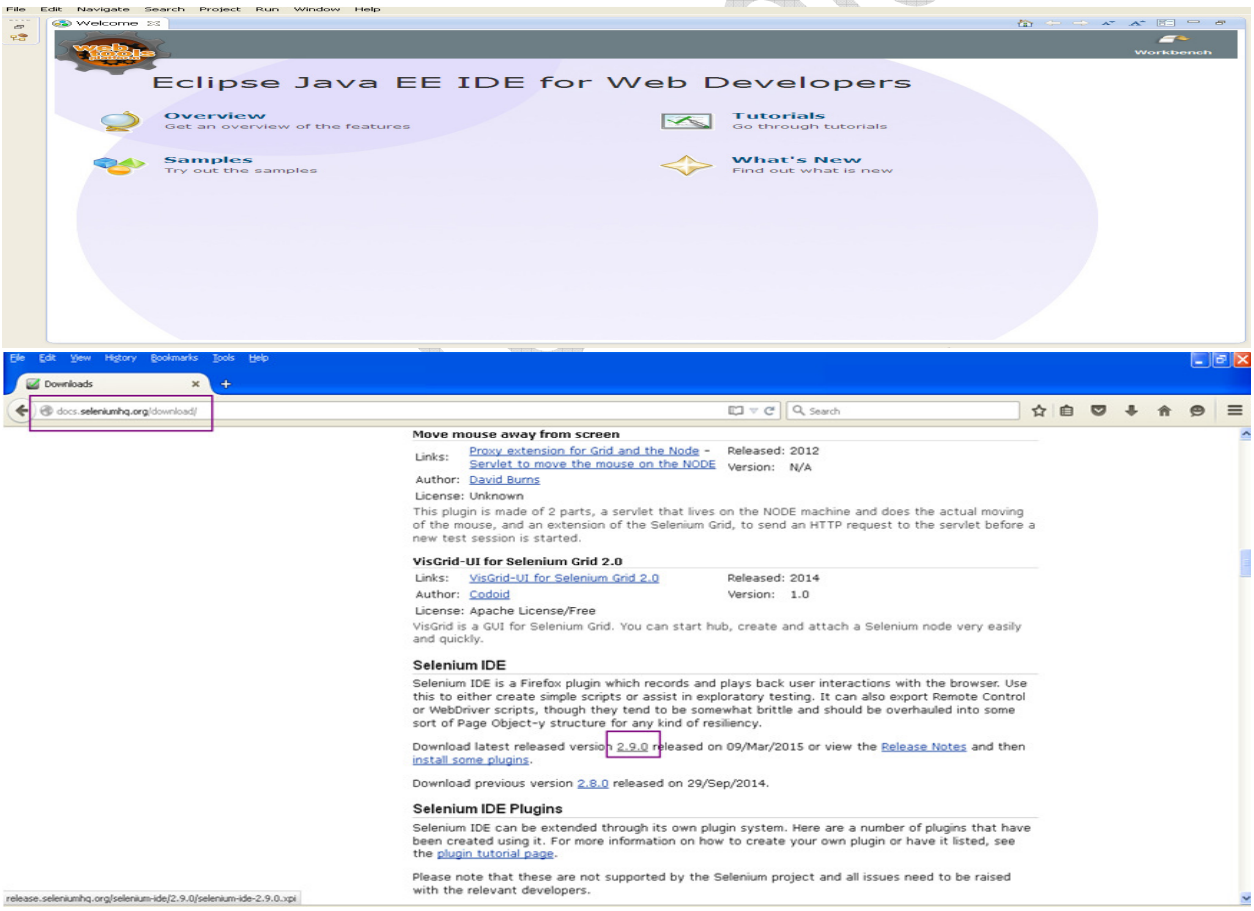
- An interface can extend multiple interfaces.

Introduction to Eclipse

- Java IDE Tools
- To write your Java programs, you will need a text editor. There are even more sophisticated IDEs available in the market.
- Notepad: On Windows machine you can use any simple text editor like Notepad, TextPad.
- Netbeans: is a Java IDE that is open-source.
- Eclipse: is also a Java IDE developed by the eclipse open-source community.

About Eclipse IDE

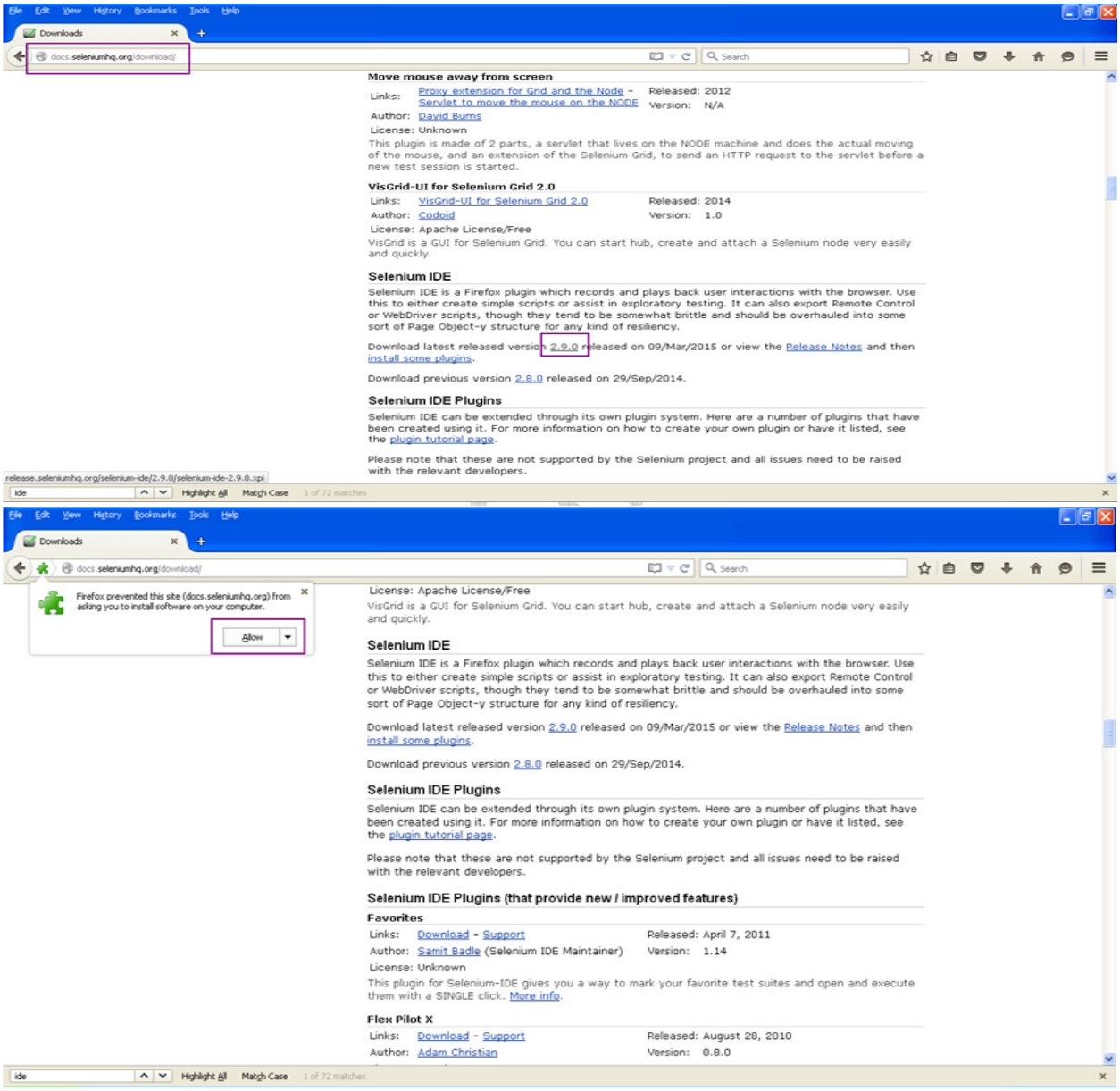
- Most people know Eclipse as an integrated development environment (IDE) for Java.
- Today it is the leading development environment for Java with a market share of approximately 65%.
- The Eclipse IDE can be extended with additional software components.
- Eclipse calls these software components plug-ins. Several Open Source projects and companies have extended the Eclipse IDE

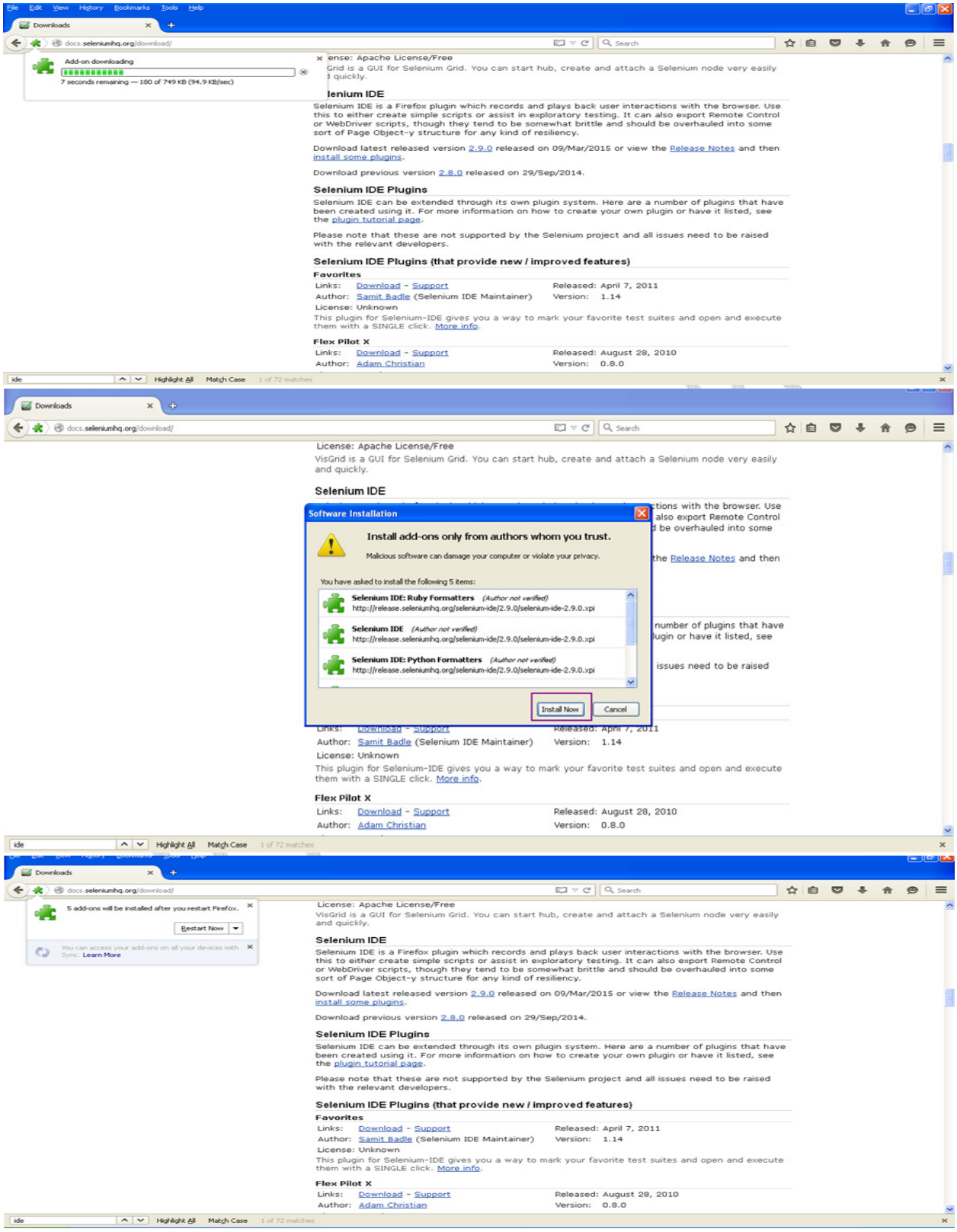


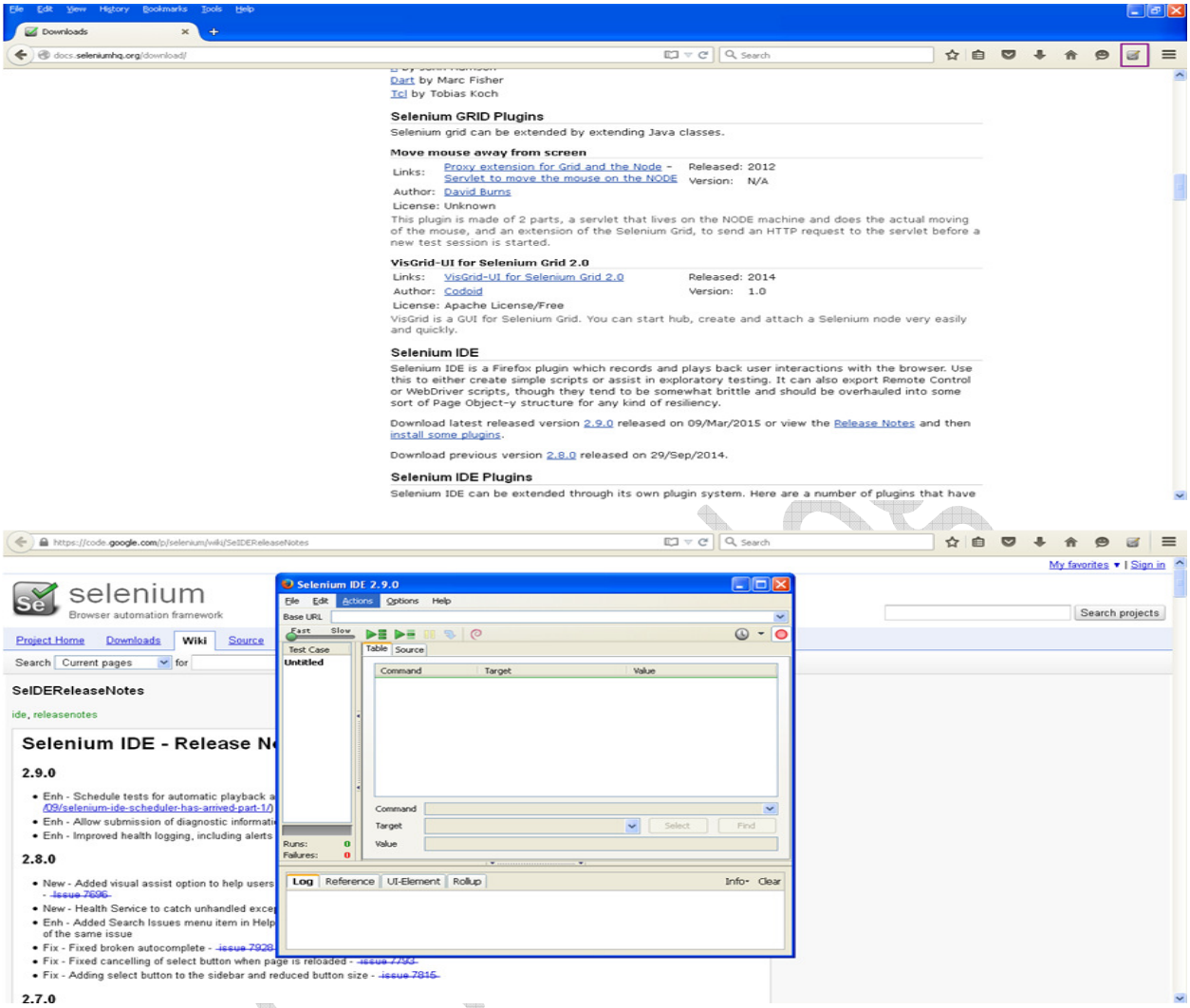
- **Selenium IDE**
  - Download and Installation
  - Record and playback techniques

- Modifying the script using IDE
- Object, CSS, XPath, Elements Indentify Process
- Validate the locator value using IDE
- Commenting Code
- Regular Expression
- Fire Bug and Fire path

Download and installation of IDE

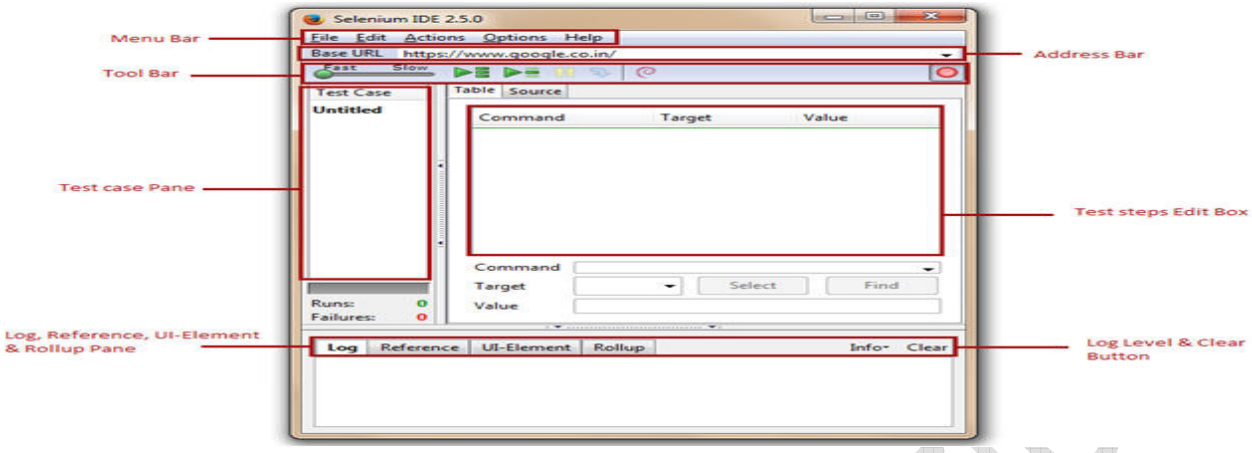



Downloads
docs.seleniumhq.org/download/
Add-on downloading
7 seconds remaining — 180 of 749 KB (94.9 KB/sec)
License: Apache License/Free
Grid is a GUI for Selenium Grid. You can start hub, create and attach a Selenium node very easily
quickly.
lenium IDE
Selenium IDE is a Firefox plugin which records and plays back user interactions with the browser. Use this to either create simple scripts or assist in exploratory testing. It can also export Remote Control or WebDriver scripts, though they tend to be somewhat brittle and should be overhauled into some sort of Page Object-y structure for any kind of resiliency.
Download latest released version 2.9.0 released on 09/Mar/2015 or view the Release Notes and then install some plugins.
Download previous version 2.8.0 released on 29/Sep/2014.
Selenium IDE Plugins
Selenium IDE can be extended through its own plugin system. Here are a number of plugins that have been created using it. For more information on how to create your own plugin or have it listed, see the plugin tutorial page.
Please note that these are not supported by the Selenium project and all issues need to be raised with the relevant developers.
Selenium IDE Plugins (that provide new / improved features)
Favorites
Links: Download - Support
Author: Samit Badle (Selenium IDE Maintainer)
Released: April 7, 2011
Version: 1.14
License: Unknown
This plugin for Selenium-IDE gives you a way to mark your favorite test suites and open and execute them with a SINGLE click. More info.
Flex Pilot X
Links: Download - Support
Author: Adam Christian
Released: August 28, 2010
Version: 0.8.0
ide
Highlight All
Match Case
1 of 72 matches
Software Installation
Install add-ons only from authors whom you trust.
Malicious software can damage your computer or violate your privacy.
You have asked to install the following 5 items:
Selenium IDE: Ruby Formatters (Author not verified)
http://release.seleniumhq.org/selenium-ide/2.9.0/selenium-ide-2.9.0.xpi
Selenium IDE (Author not verified)
http://release.seleniumhq.org/selenium-ide/2.9.0/selenium-ide-2.9.0.xpi
Selenium IDE: Python Formatters (Author not verified)
http://release.seleniumhq.org/selenium-ide/2.9.0/selenium-ide-2.9.0.xpi
Install Now
Cancel
5 add-ons will be installed after you restart Firefox.
Restart Now
You can access your add-ons on all your devices with Sync. Learn More



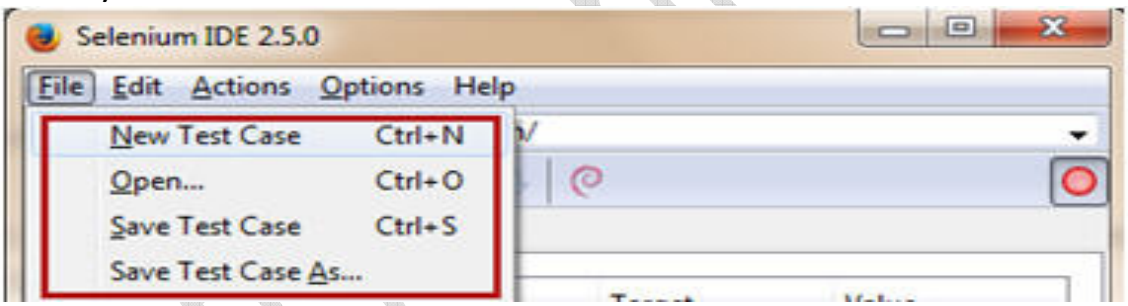
Features of Selenium IDE

- Let's have a look at each of the feature in detail.



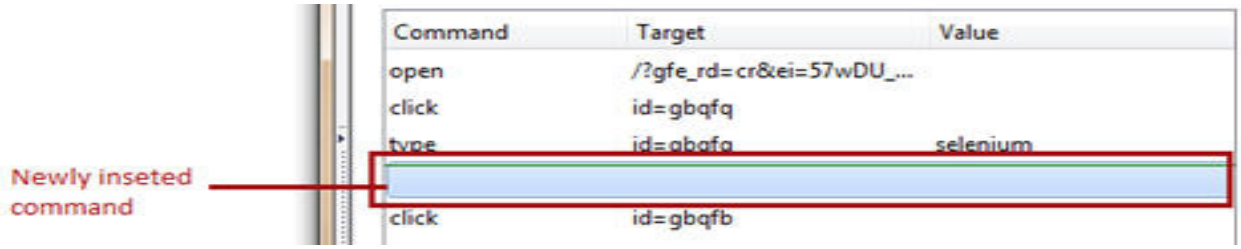
Menu Bar

- Menu bar is positioned at the upper most of the Selenium IDE window. The menu bar is typically comprised of five modules.
    - File Menu
    - Edit Menu
    - Actions Menu
    - Options Menu
    - Help Menu
- **A) File Menu**



- File Menu is very much analogous to the file menu belonging to any other application. It allows user to:
    - Create new test case, open existing test case, save the current test case.
    - Export Test Case As and Export Test Suite As in any of the associated programming language compatible with Selenium RC and WebDriver. It also gives the liberty to the user to prefer amid the available unit testing frameworks like jUnit, TestNG etc.

Thus an IDE test case can be exported for a chosen union of programming language, unit testing framework and tool from the selenium package



- Now the user can insert the actual command action, target and value.

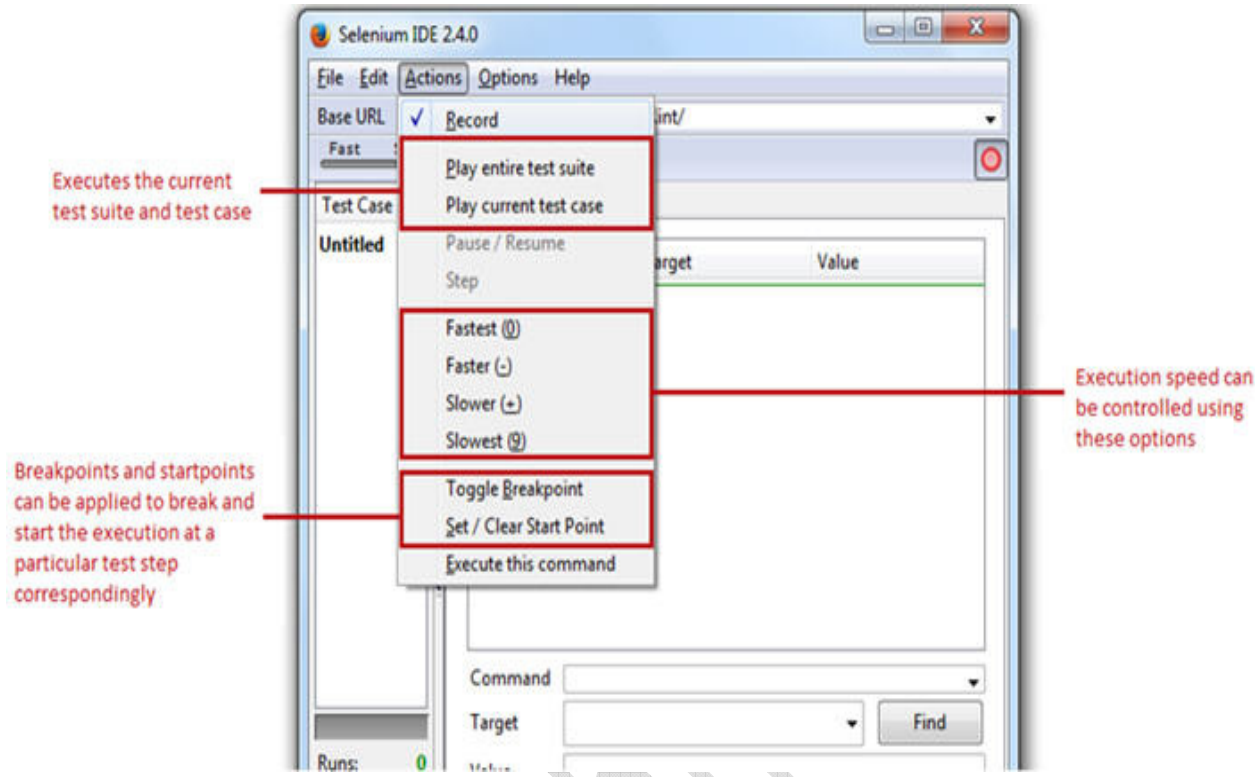
Command	Target	Value
open	/?gfe_rd=cr&ei=57wDU_...	
click	id=gbqfq	
type	id=gbqfq	selenium
typeAndWait	id=test	Newly inserted command
click	id=gbqfb	

- **Insert New Comment**
  - In the same way we can insert comments.

Command	Target	Value
open	/?gfe_rd=cr&ei=57wDU_...	
click	id=gbqfq	
type	id=gbqfq	selenium
typeAndWait	id=test	Newly inserted command
Inserting new com...		
click	id=gbqfb	

The purple color indicates that the text is representing a comment.

- **C) Actions Menu**



Actions Menu equips the user with the options like:

Record – Record options fine tunes the Selenium IDE into the recording mode. Thus, any action made by the user on the Firefox browser would be recorded in IDE.

Play entire test suite – The option plays all the Selenium IDE test cases associated with the current test suite.

Play current test case – The option plays the current Selenium IDE test case that has been recorded/created by the user.

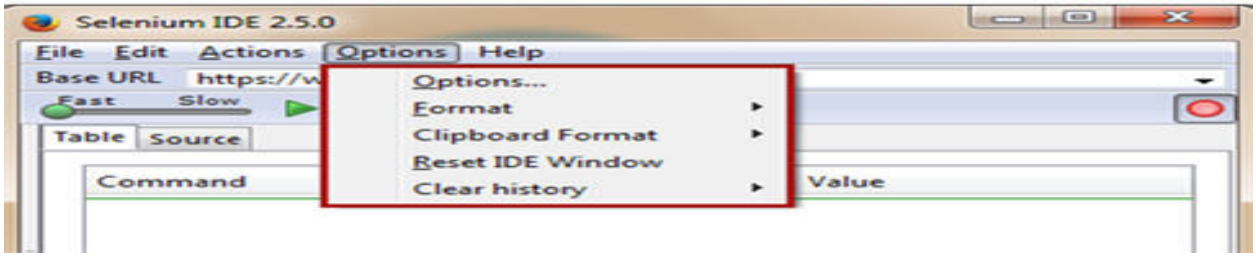
Pause / Resume – User can Pause/Resume the test case at any point of time while execution.

Toggle Breakpoint – User can set one or multiple breakpoint(s) to forcefully break the execution at any particular test step during execution.

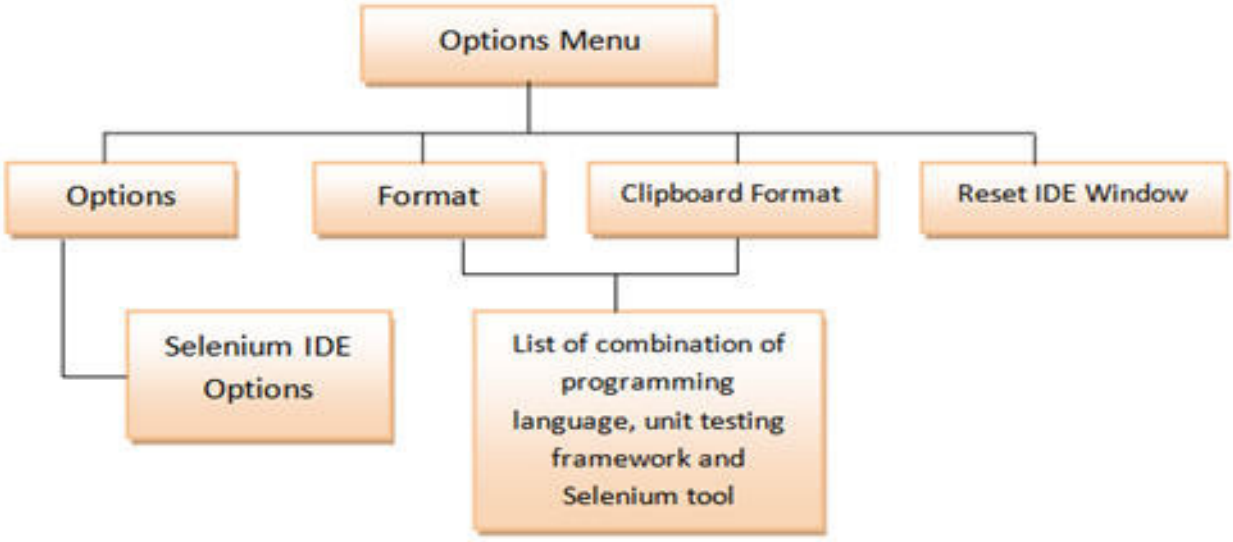
Set / Clear Start Point – User can also set start point at any particular test step for execution. This would enable user to execute the test case from the given start point for the subsequent runs.

To deal with the page/element loads, the user can set the execution speed from fastest to lowest with respect to the responsiveness of the application under test.

- **D) Options Menu**



Options menu privileges the user to set and practice various settings provided by the Selenium IDE. Options menu is recommended as one of the most important and advantageous menu of the tool. Options Menu is primarily comprised of the following four components which can be sub-divided into the following:



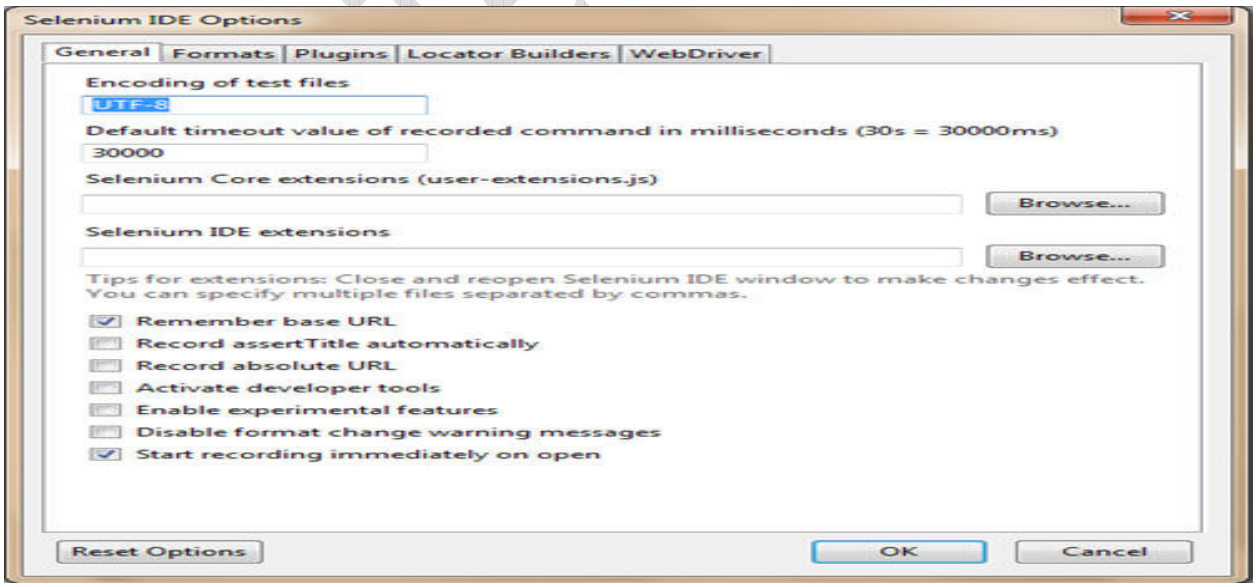
- **Options**

Selenium IDE Options dialog box

To launch Selenium IDE Options dialog box, follow the steps:

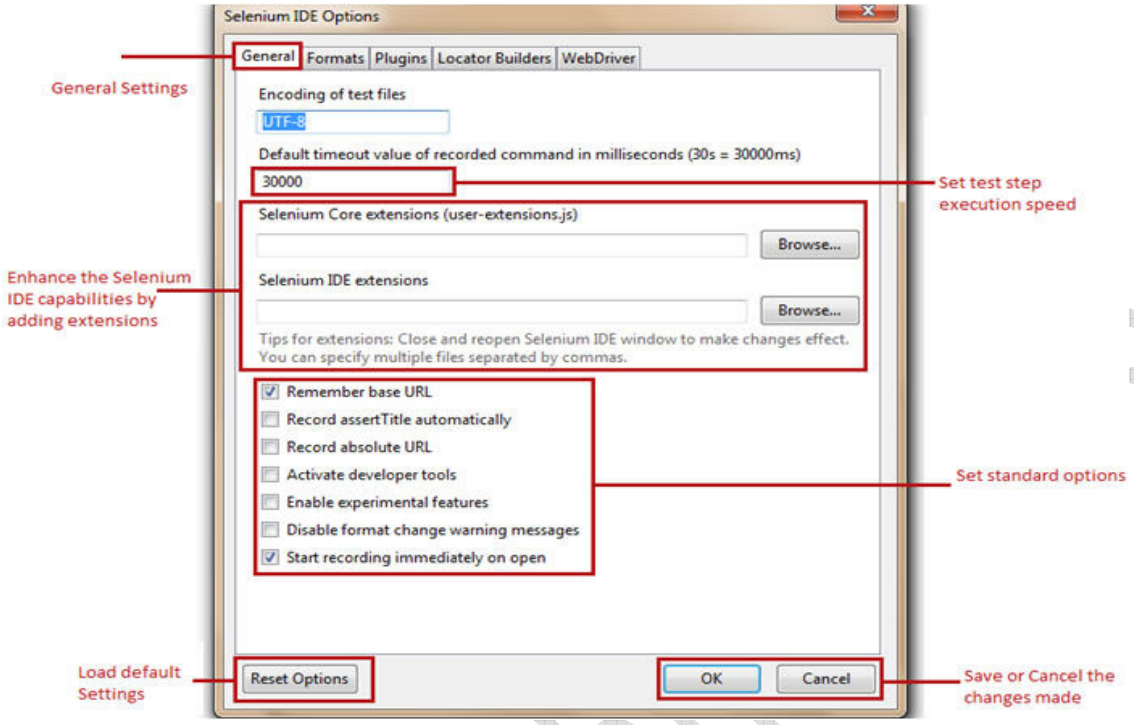
- Click on Options Menu
- Click on the Options

A Selenium IDE Options dialog box appears. Refer the following figure.



Selenium IDE Options dialog box aids the user to play with the general settings, available formats, available plug-ins and available locators types and their builders. Let's have a look at the few important ones.

- **General Settings**

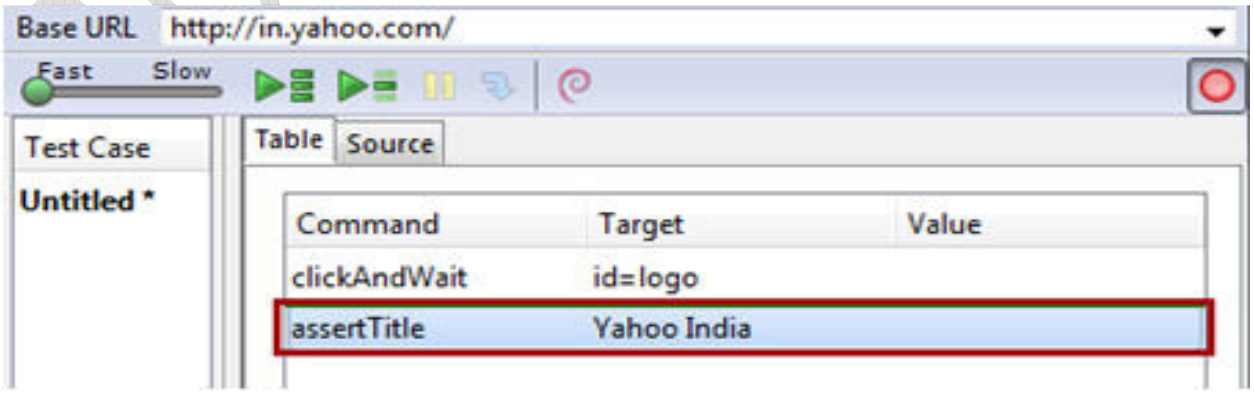


Default Timeout Value – Default Timeout Value represents the time (in milliseconds) that selenium would wait for a test step to execute before generating an error. The standard timeout value is 30000 milliseconds i.e. 30 seconds. The user can leverage this feature by changing the default time in cases when the web element takes more/less than the specified time to load.

Extensions – Selenium IDE supports a wide range of extensions to enhance the capabilities of the core tool thereby multiplying its potential. These user extensions are simply the JavaScript files. They can set by mentioning their absolute path in the text boxes representing extensions in the Options dialog box.

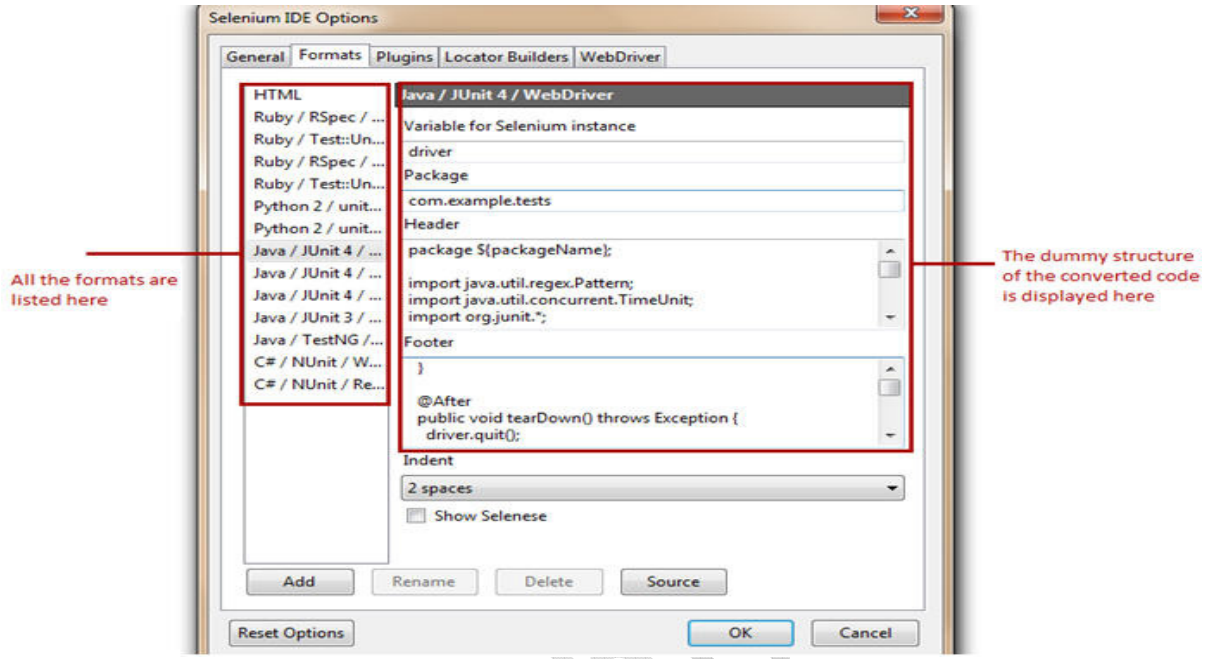
Remember base URL – Checking this option enables the Selenium IDE to remember the URL every time we launch it. Thus it is advisable to mark it checked. Un-checking this option will leave the base URL field as blank and it will be re-filled only when we launch another URL on the browser.

Record assert Title automatically – Checking this field inserts the assert Title command automatically along with the target value for every visited web page.



Enable experimental features – Checking this field for the first time imports the various available formats into the Selenium IDE.

Formates:



Formats tab displays all the available formats with selenium IDE. User is levied with the choice to enable and disable any of the formats. Refer the following figure.

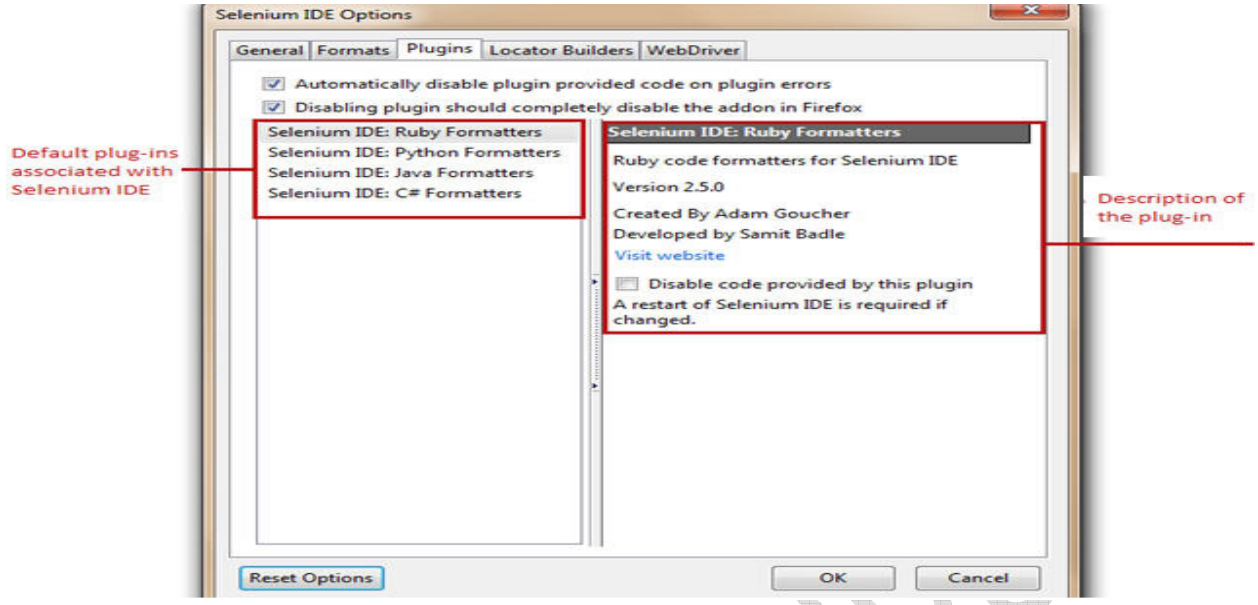
Selenium IDE Plugins

Plug-ins tab displays the supported Firefox plug-ins installed on our instance of Selenium IDE. There are a number of plug-ins available to cater different needs, thus we can install these add-ons like we do other plug-ins. One of the recently introduced plug-in is “File Logging”. In the end of this tutorial, we will witness how to install and use this plug-in.

With the standard distribution, Selenium IDE comes with a cluster of following plug-ins:

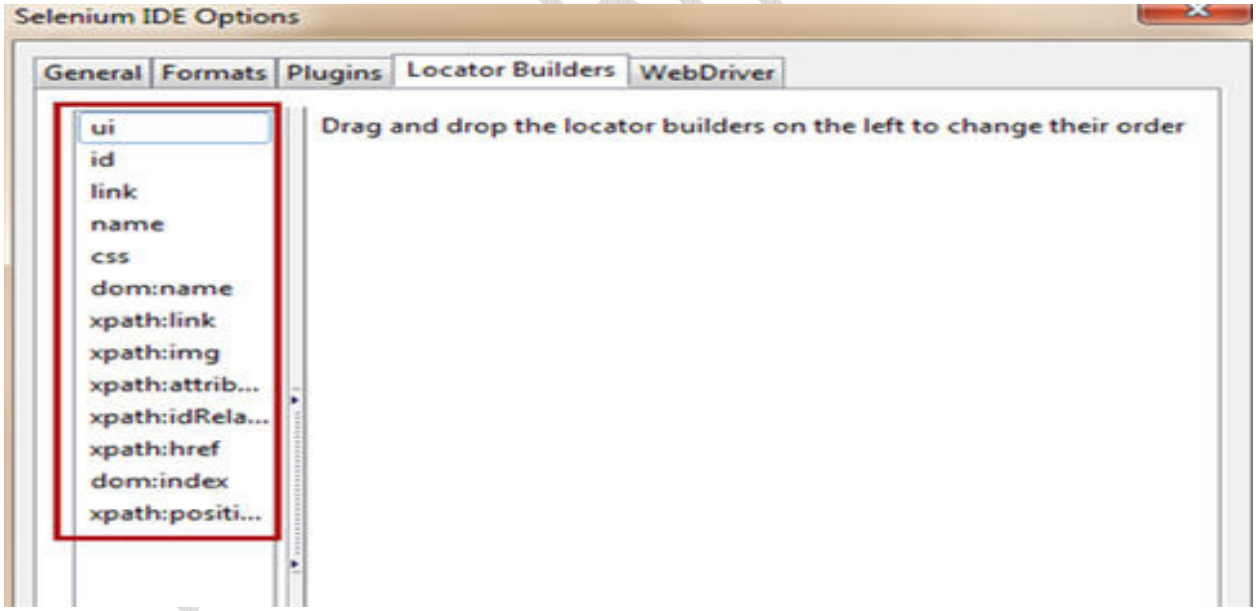
- Selenium IDE: Ruby Formatters
- Selenium IDE: Python Formatters
- Selenium IDE: Java Formatters
- Selenium IDE: C# Formatters

These formatters are responsible to convert the HTML test cases into the desired programming formats**.**



Locator Builders

Locator builders allow us to prioritize the order of locator types that are generated while recording the user actions. Locators are the set of standards by which we uniquely identify a web element on a web page



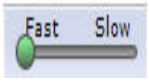
Now, whenever the user uses “open” command of Selenium IDE without a target value, the base URL would be launched on to the browser.

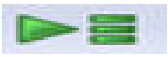
Accessing relative paths

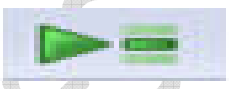
To access relative paths, user simply needs to enter a target value like “/download” along with the “open” command. Thus, the base URL appended with “/downloads” (http://docs.seleniumhq.org/resources) would be launched on to the browser. The same is evident in the above depiction.

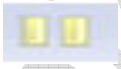
Toolbar: 

Toolbar provides us varied options pertinent to the recording and execution of the test case

Playback Speed – This option allows user to control the test case execution speed from fast to slow. 

Play test suite – This option allows user to execute all the test cases belonging to the current test suite sequentially. 

Play test case – This option allows user to execute the currently selected test case 

Pause – This option allows user to pause the current execution. 

Step – This option allows user to step into the test step. 

Rollup-This option allows user to combine multiple test steps to act like a single command. 

Record: This option allows user to start/stop the recording of user actions. The hollow red ball indicates the start of the recording session whereas the solid red ball indicates the end of the recording session. By default, the Selenium IDE opens in the recording mode.

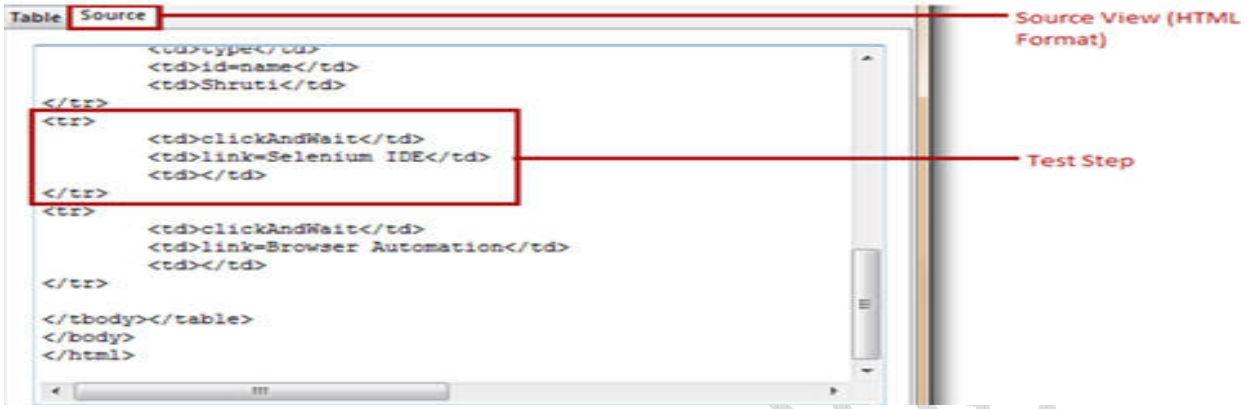
Editor: Editor is a section where IDE records a test case. Each and every user action is recorded in the editor in the same order in which they are performed.

- The editor in IDE has two views, namely:
    - 1) Table View:

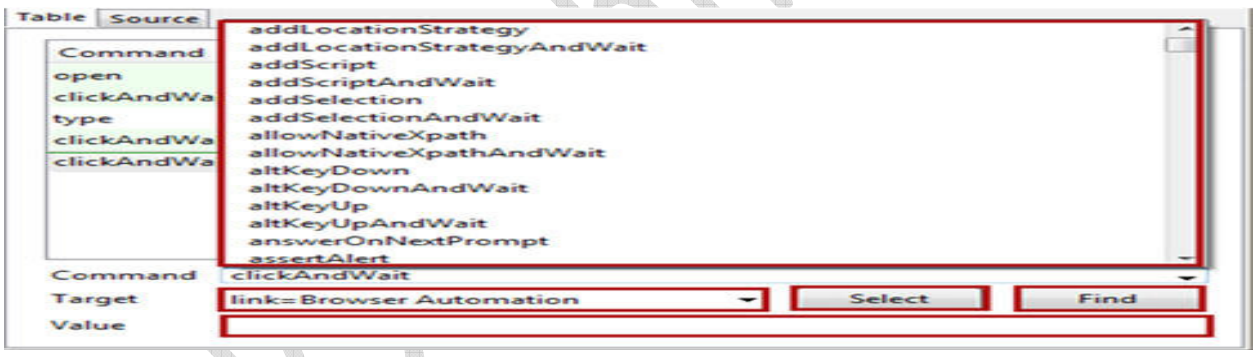
It is the default view provided by Selenium IDE. The test case is represented in the tabular format. Each user action in the table view is a consolidation of “Command”, “Target” and “Value” where command, target and value refers to user action, web element with the unique identification and test data correspondingly. Besides recording it also allows user to insert, create and edit new Selenese commands with the help of the editor form present in the bottom.

2) Source View

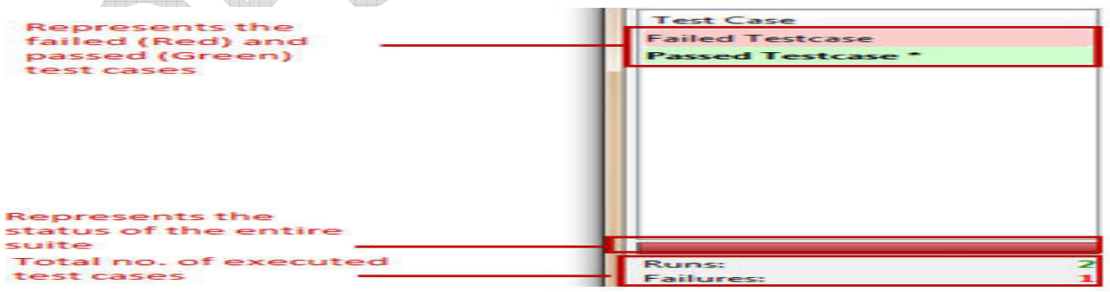
The test case is represented in the HTML format. Each test step is considered be a row <tr> which is a combination of command, target and value in the separate columns <td>.
Like any HTML document, more rows and columns can be added to correspond to each Selenese command.



Editor Form lets the user to type any command and the suggestions for the related command would be populated automatically.Select button lets the user to select any web element and its locator would be fetched automatically into the target field. Find button lets the user find the web element on the web page against a defined target. Value is the test input data entered into the targets with which we want to test the scenario.



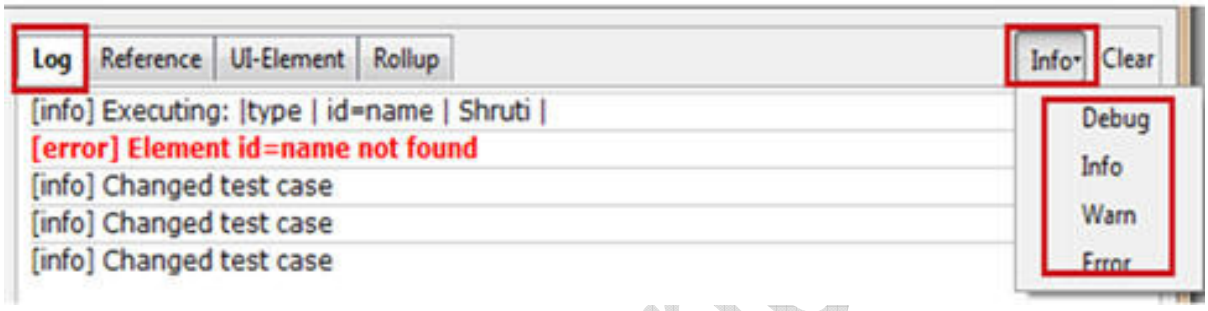
- **Test case pane**



At the instance we open Selenium IDE interface, we see a left container titled “Test case” containing an untitled test case. Thus, this left container is entitled as Test case pane.
Test case pane contains all the test cases that are recorded by IDE. The tool has a capability of opening more than one test case at the same time under test case pane and the user can easily shuffle between the test cases.

- The test steps of these test cases are organized in the editor section.
- Selenium IDE has a color coding ingredient for reporting purpose. After the execution, the test case in marked either in “red” or “green” color.
- Red color symbolizes the unsuccessful run i.e. failure of the test case.
- Green color symbolizes the successful run of the test case
- It also layouts the summary of the total number of test cases executed with the number of failed test cases.
- If we execute a test suite, all the associated test cases would be listed in the test case pane. Upon execution, the above color codes would be rendered accordingly.

Log Pane:



Log pane gives the insight about current execution in the form of messages along with the log level in the real time. Thus, log messages enable a user to debug the issues in case of test case execution failures.

The printing methods / log levels used for generating logs are:

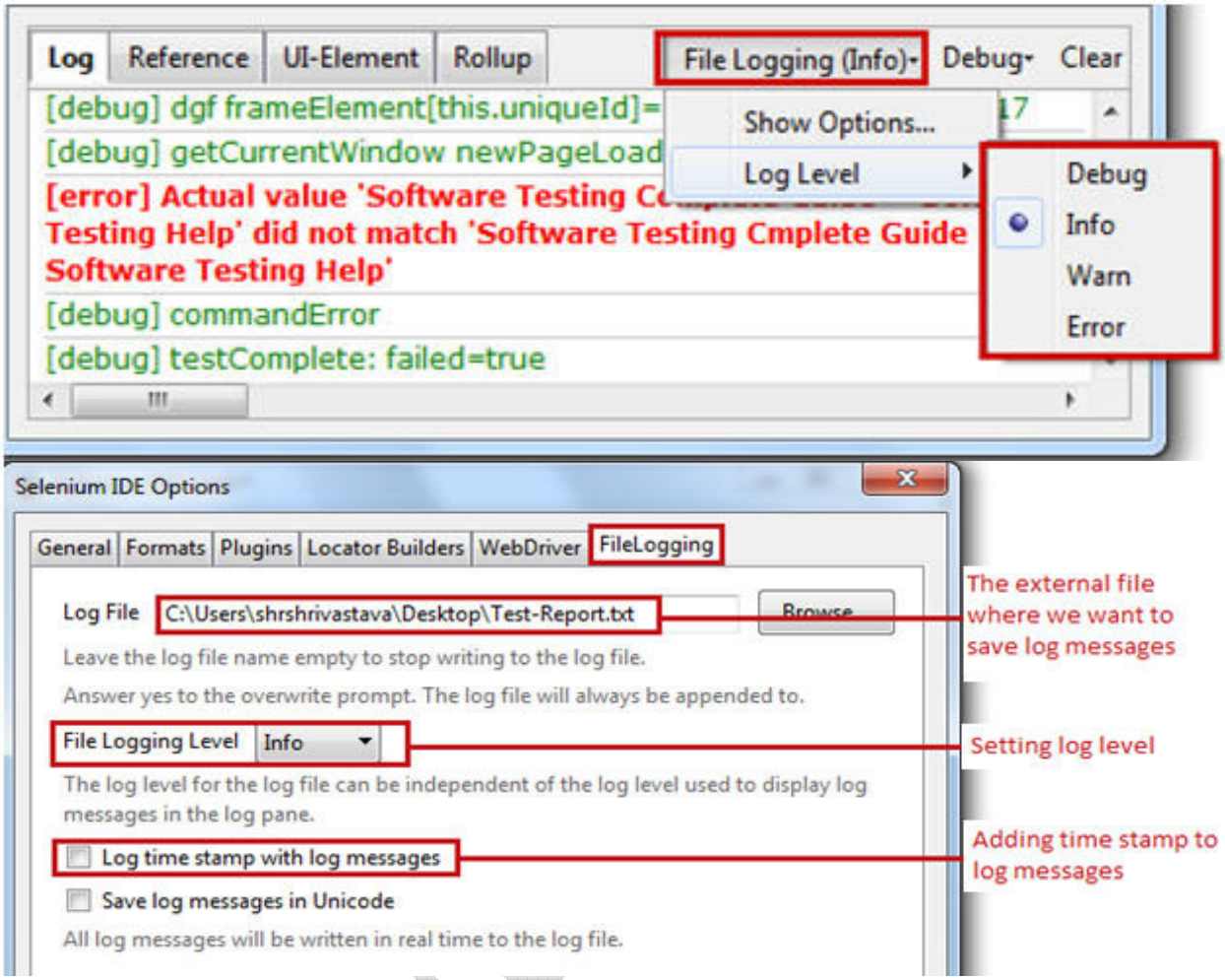
Error – Error message gives information about the test step failure. It may be generated in the cases when element is not found, page is not loaded, verification/assertion fails etc.

- Warn – Warning message gives information about unexpected conditions.
- Info – Info message gives information about current test step execution.
- Debug – Debug messages gives information about the technicalities in the backdrop about the current test step.

Logs can be filtered with the help of a drop down located at the top-right corner of the footer beside the clear button. Clear button erases all the log messages generated in the current or previous run.

- **Generating Logs in an external medium**

Recently introduced “File Logging” plug-in enables the user to save log messages into an external file. File Logging can be plugged in to IDE like any other plug-in. Upon installation, it can be found as a tab named “File Logging” in the footer beside the Clear button



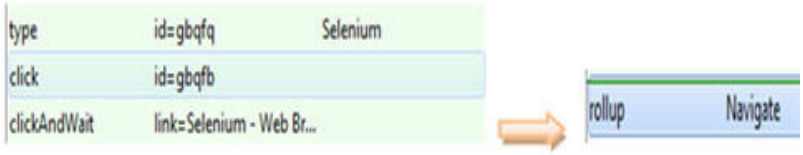
Reference Pane

Reference Pane gives the brief description about the currently selected Selenese command along with its argument details.

UI-Element Pane

UI – Element Pane allows Selenium user to use JavaScript Object Notation acronym as JSON to access the page elements. More on this can be found in UI-Element Documentation under Help Menu.

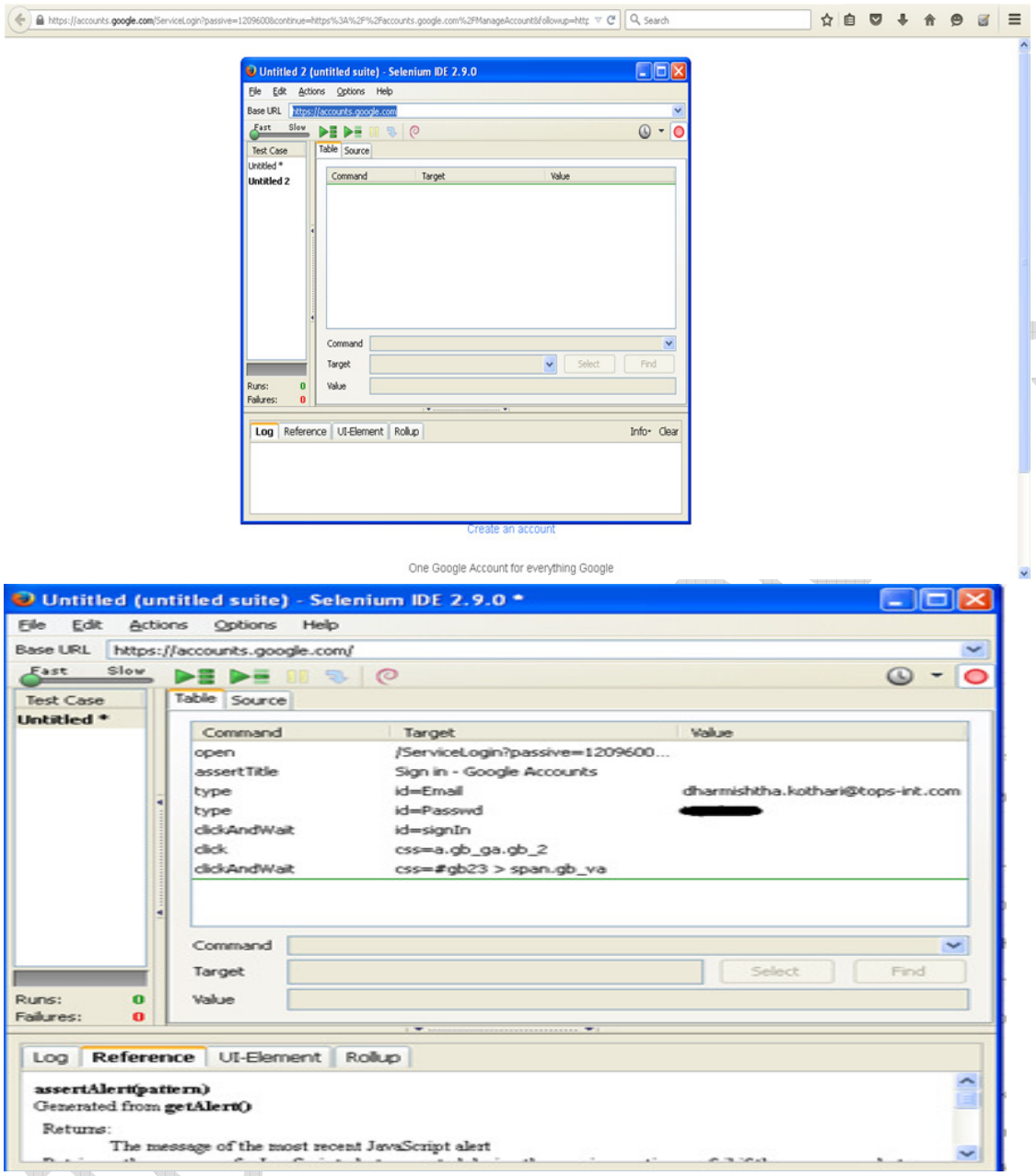
Rollup Pane



Rollup Pane allows the user to roll up or combine multiple test steps to constitute a single command termed as "rollup". The rollup in turn can be called multiple times across the test case.

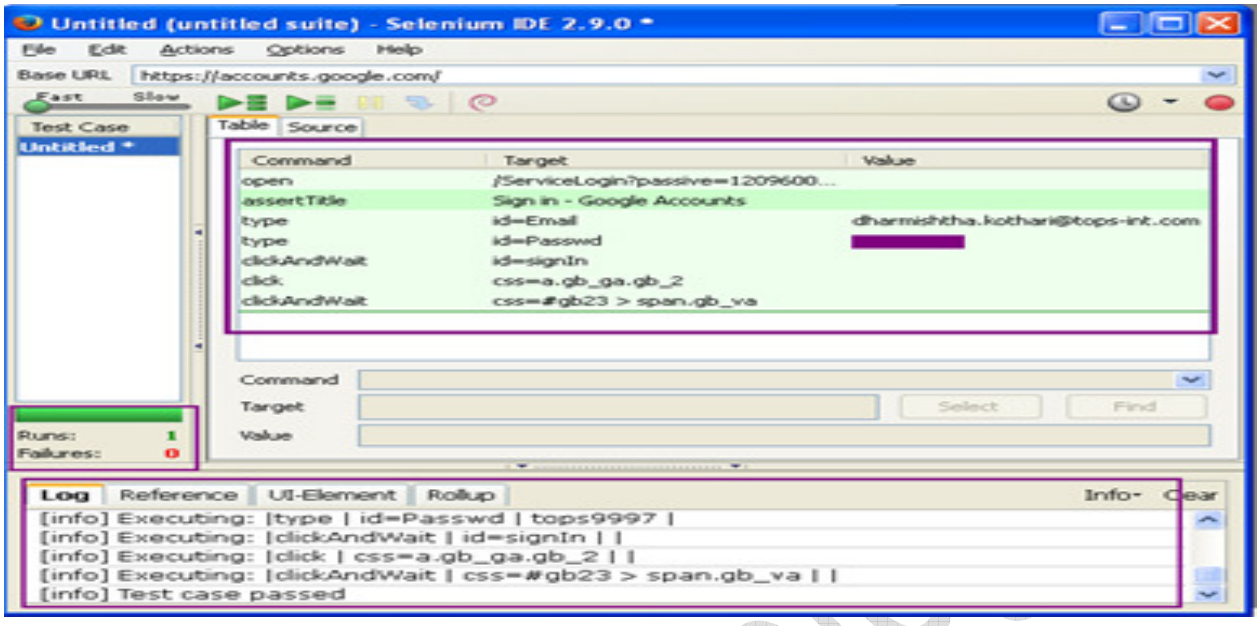
Record and Playback Techniques

- **Creating First Selenium IDE Script**
  - So let us now create our first script using Selenium IDE.
- **The entire script creation process can be classified into 3 chunks:**
- **Process #1: Recording** – Selenium IDE aids the user to record user interactions with the browser and thus the recorded actions as a whole are termed as Selenium IDE script.
- **Process #2: Playing back** – In this section, we execute the recorded script so as to verify and monitor its stability and success rate.
- **Process #3: Saving** – Once we have recorded a stable script, we may want to save it for future runs and regressions.
- Let us now see their implementation.
- **Process #1: Recording a test script**
- **Scenario**
- Open "https://accounts.google.com".
- Assert Title of the application
- Enter a valid username and password and submit the details to login.
- Verify that the user is re-directed to the Home page.
- **Step 1 –** Launch the Firefox and open Selenium IDE from the menu bar.
- **Step 2 –** Enter the address of application under test ("https://accounts.google.com") inside the Base URL textbox.



Process # 2: Playing back / executing a test script

Now that we have created our first Selenium IDE script, we would want to execute it to see if the script is stable enough. Click on the playback button to execute the script.



Process #3: Saving a test script

Once, we have played back the script, now it's time to save the created test script.

Step 1 – To save the test script, Click on the File menu and select "Save Test Case" option.

Step 2 – The system will prompt us to browse or enter the desired location to save our test case and to provide the test script name. Furnish the test name as "Gmail_Login" and click on the "Save" button. The test script can be found at the location provided in the above step. Notice that the test script is saved in HTML format.

Using Common features of Selenium IDE

Setting Execution speed

While testing web applications, we come across several scenarios where an action performed may trigger a page load. Thus we must be cognizant enough while dealing such scenarios. So to avoid failures while playing back these test scenarios, we can set the execution speed to be minimal. Refer the following figure for the same.

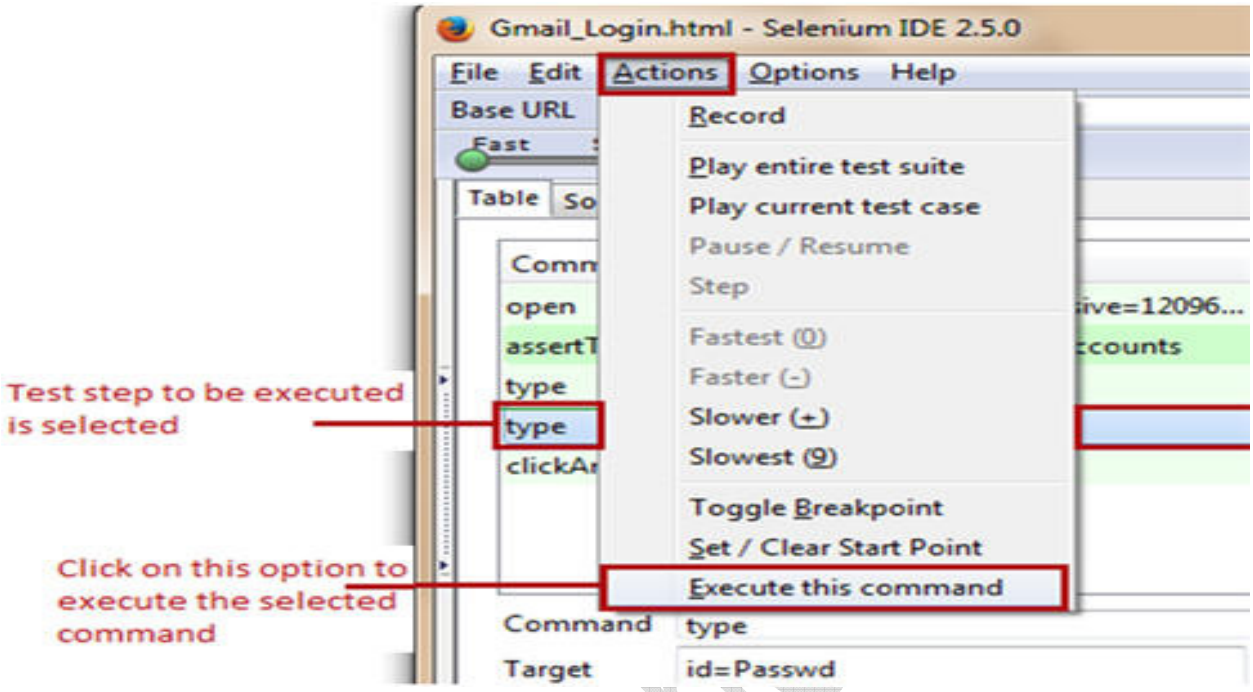
Using "Execute this command" option

Selenium IDE allows the user to execute a single test step within the entire test script without executing the entire test script. "Execute this command" is the option which makes this obtainable.

"Execute this command" option can be used at times when we want to debug/see the behavior of a particular test step.

"Execute this command" option can be used in the following four ways:

- **#1.** Using Actions tab from the Menu bar



#2. Using short cut key ("X")

#3. Right click the test step and select "Execute this command"

#4. Double click the test step

In all the above cases, user is expected to select the test step which he / she want to execute.

Steps to be followed:

Step 1 – Launch the web browser and open the target URL ("https://accounts.google.com"), Select the test step that we desire to execute. Remember to open correct web page to mitigate the chances of failure.

Step 2 – Press "X" to execute the selected test step. Alternatively, one can use other ways too.

Step 3 – Notice that the selected test step is executed. The test step would be color coded in green for success or red for failure. At the same time, the test step would be simulated into an action on the web browser.

Note that the user is responsible to bring the script before executing the test step and Firefox in context. There is a probability of failure if the user has not opened the legitimate web page.

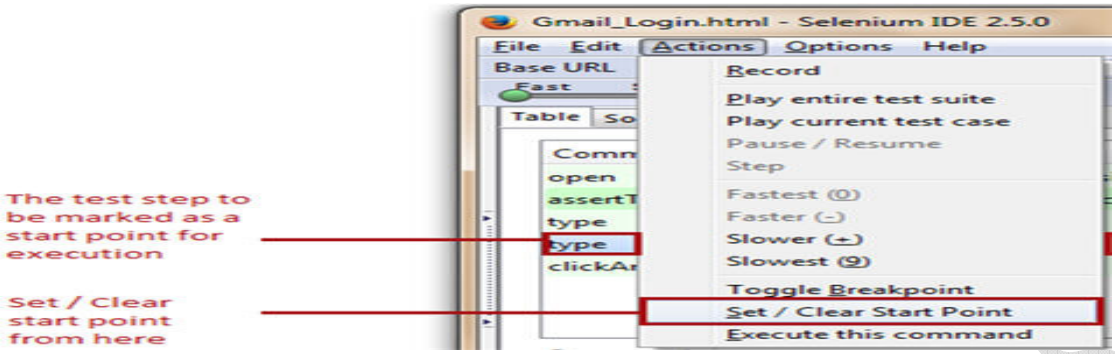
Using Start point

Selenium IDE allows the user to specify a start point within a test script. The start point points to the test step from where we wish to start the test script execution.

Start point can be used at times when we do not desire to execute the entire test script starting from the beginning rather we customize the script to execute from a certain step.

Start point can be set and clear in the following three ways:

- **#1.** Using Actions tab from the Menu bar
- **#2.** Using short cut key ("S")



#3. Right click the test step and select "Set/Clear Start Point". Menu similar to above image will be displayed. In all the above cases, user is expected to select the test step from where he wants to start the execution prior to setting start point. As soon as the user has marked the test step to indicate the start point, an icon gets affixed to it.

Notes

There can be one and only one start point in a single script.

The start point can be cleared in the same way it was set.

User is responsible to bring the script after applying start point and Firefox in context. There is a probability of failure if the user has not opened the legitimate web page.

Using Break point

Selenium IDE allows the user to specify break points within a test script. The break points indicate Selenium IDE where to pause the test script.

Break points can be used at times when we desire to break the execution in smaller logical chunks to witness the execution trends.

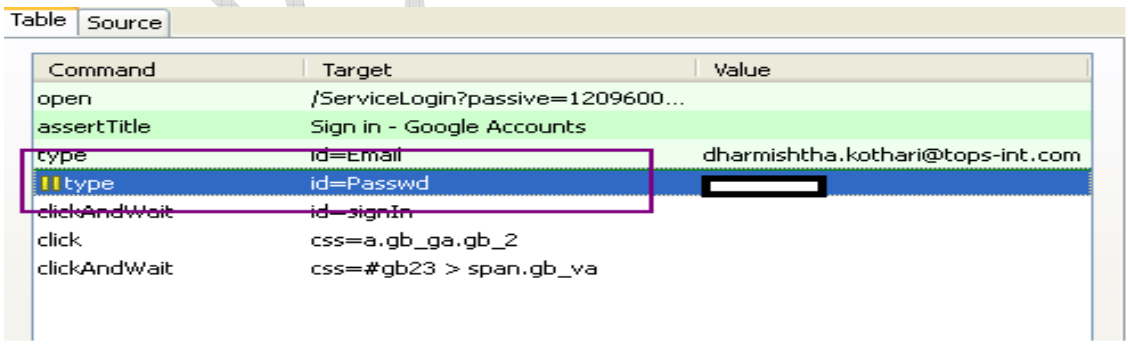
Break point can be set and clear in the following three ways:

Using Actions tab from the Menu bar

Right click the test step and select "Toggle Breakpoint".

Using short cut key ("B")

As soon as the user has marked the test step to indicate the break point, an icon gets affixed to it.



- **Apply multiple breakpoints**
  - Selenium IDE allows user to apply multiple breakpoints in a single test script. Once the first section of the test script is executed, the script pauses as and when the breakpoint

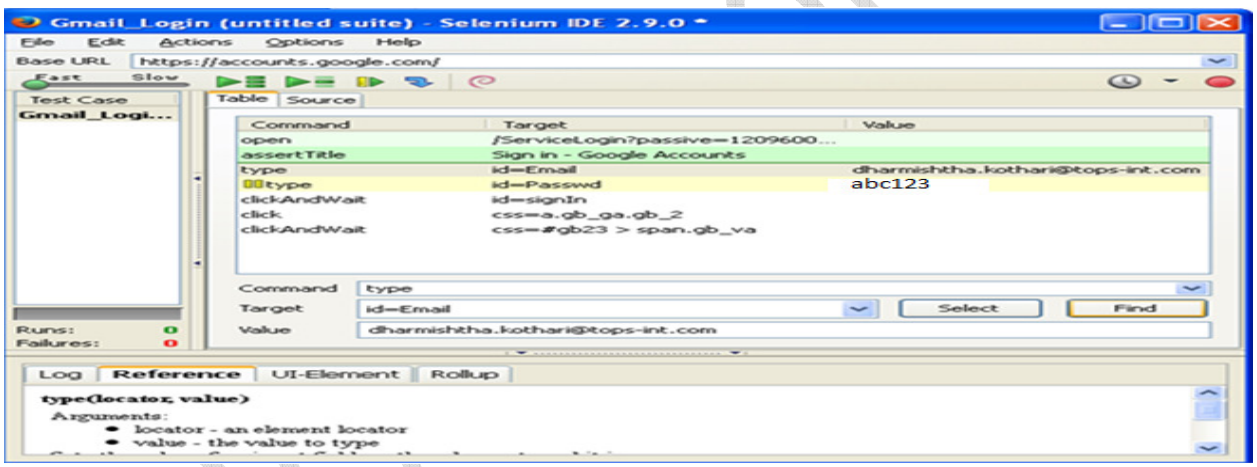
is reached. To execute the subsequent test steps, user is required to execute each of the test steps explicitly.

- **Notes**
  - There can be as many break points as you wish in a single script.
  - The break point can be cleared in the same way it was set.
- **Using Find Button**

One of the most crucial aspects of Selenium IDE test scripts is to find and locate web elements within a web page. At times, there are web elements which have analogous properties associated with them, thus making it challenging for a user to identify a particular web element uniquely.

To address this issue, Selenium IDE provides Find button. The Find Button is used to ascertain that locator value provided in the Target test box is indeed correct and identifies the designated web element on the GUI. Let us consider the above created Selenium IDE test script. Select any command and notice the target text box. Click on the Find button present just beside the Target text box.

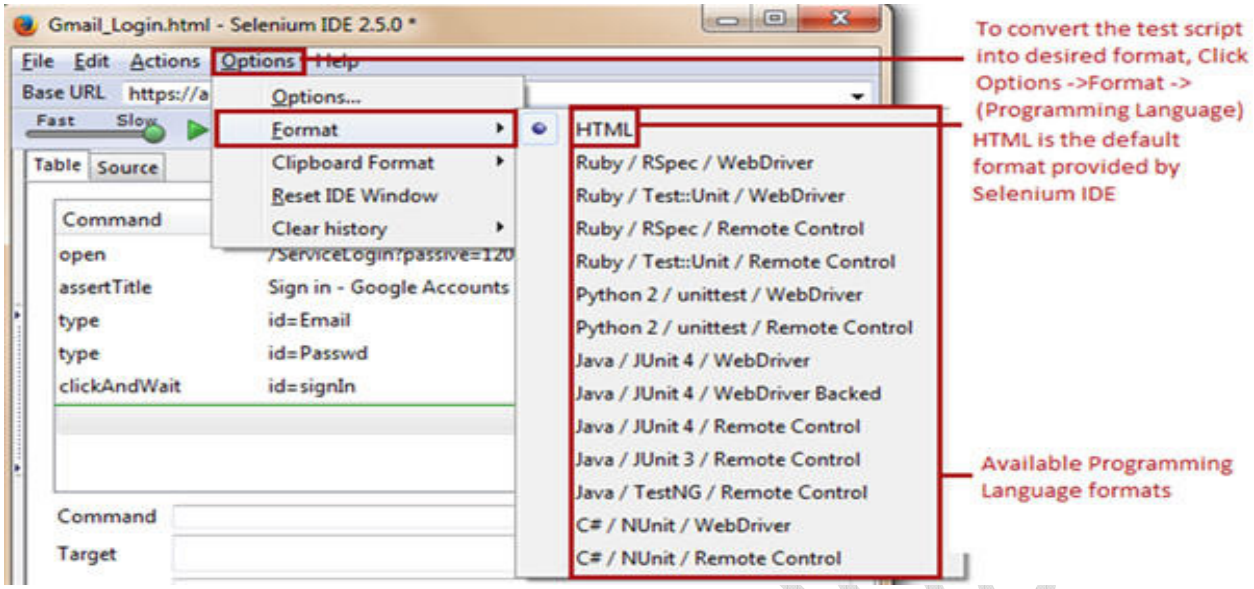
Notice that the corresponding web element would be highlighted in yellow with a fluorescent green border around it. If no or wrong web element is highlighted, then the user is required to rectify the issue and would need to impose some other locator value.



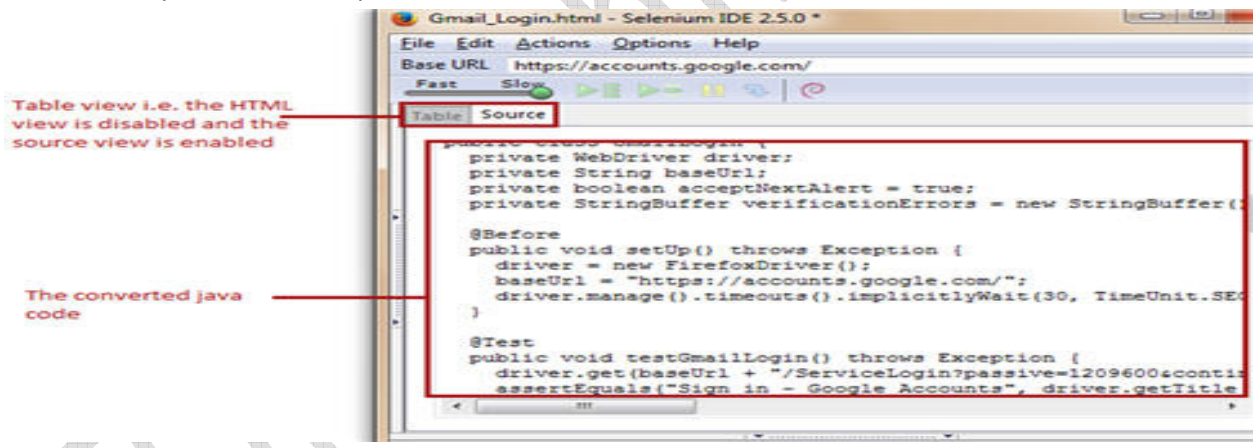
- **Using Other Formats**

Converting Selenium IDE test scripts to Other Programming Languages. Selenium IDE supports conversion test scripts into set of programming languages from a default type (HTML). The converted test scripts cannot be played back using Selenium IDE until and unless it is reverted back to HTML. Thus the conversion is beneficial and constructive only when we are executing it from other tools of Selenium Suite.

Step 1 – Click on the options tab under the menu bar and select the programming language format under format option in order to convert the test script into our desired format

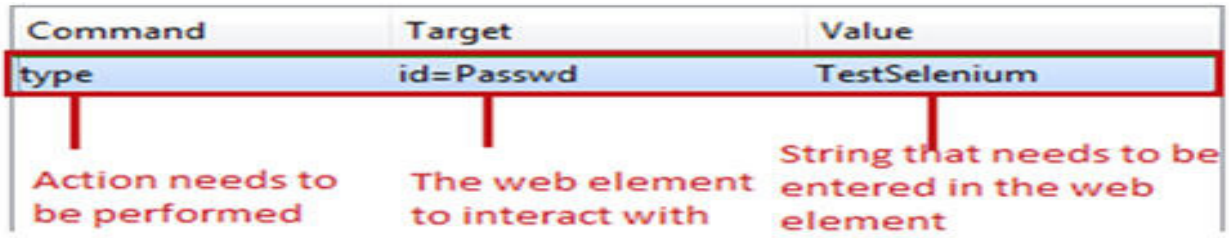


Step 2 – As soon as we select our Desired Programming language format ("Java / JUnit4 / WebDriver" in our case), a prompt box appears which says "Changing format is now marked experimental! If you continue, recording and playback may not work, your changes may be lost and you may have to copy and paste the test in a text editor to save. It is better to make a copy of your test cases before you continue. Do you still want to proceed?" Click "OK" to continue.



Selenium IDE Commands

- Each Selenium IDE test step can chiefly be split into following three components:
  - Command
  - Target
  - Value



- **Types of Selenium IDE commands:**
- There are three flavors of Selenium IDE commands. Each of the test step in Selenium IDE falls under any of the following category.
    - Actions
    - Accessors
    - Assertions
- **Actions:**

Actions are those commands which interact directly with the application by either altering its state or by pouring some test data. For Example, "type" command lets the user to interact directly with the web elements like text box. It allows them to enter a specific value in the text box and as when the value is entered; it is showed on the UI as well. Another example is "click" command. "click" command lets the user to manipulate with the state of the application. In case of failure of an action type command, the test script execution halts and rest of the test steps would not be executed.

Accessors

Accessors are those commands which allows user to store certain values to a user defined variable. These stored values can be later on used to create assertions and verifications. For example, "storeAllLinks" reads and stores all the hyperlinks available within a web page into a user defined variable. Remember the variable is of array type if there are multiple values to store.

Assertions

Assertions are very similar to Accessors as they do not interact with the application directly. Assertions are used to verify the current state of the application with an expected state.

- **Forms of Assertions:**

assert – the "assert" command makes sure that the test execution is terminated in case of failure.
verify – the "verify" command lets the Selenium IDE to carry on with the test script execution even if the verification is failed.
waitFor – the "waitFor" command waits for a certain condition to be met before executing the next test step. The conditions are like page to be loaded, element to be present. It allows the test execution to proceed even if the condition is not met within the stipulated waiting period.

Commonly used Selenium IDE Commands:

Command	Description	Arguments
open	Opens a specified URL in the browser.	1

assertTitle, VerifyTitle	Returns the current page title and compares it with the specified title	1
assertElementPresent, verifyElementPresent	Verify / Asserts the presence of an element on a web page.	1
type, typeKeys, sendKeys	Enters a value (String) in the specified web element.	2
Click, clickAt, clickAndWait	Clicks on a specified web element within a web page.	1
waitForPageToLoad	Sleeps the execution and waits until the page is loaded completely.	1
waitForElement Present	Sleeps the execution and waits until the specified element is present	1
chooseOkOnNext Confirmation, chooseCancelOn NextConfirmation	Click on ”OK” or “Cancel” button when next confirmation box appears.	0

Introduction to Firebug:

Firebug is a Mozilla Firefox add-on. This tool helps us in identifying or to be more particular inspecting HTML, CSS and JavaScript elements on a web page. It helps us identifying the elements uniquely on a webpage.

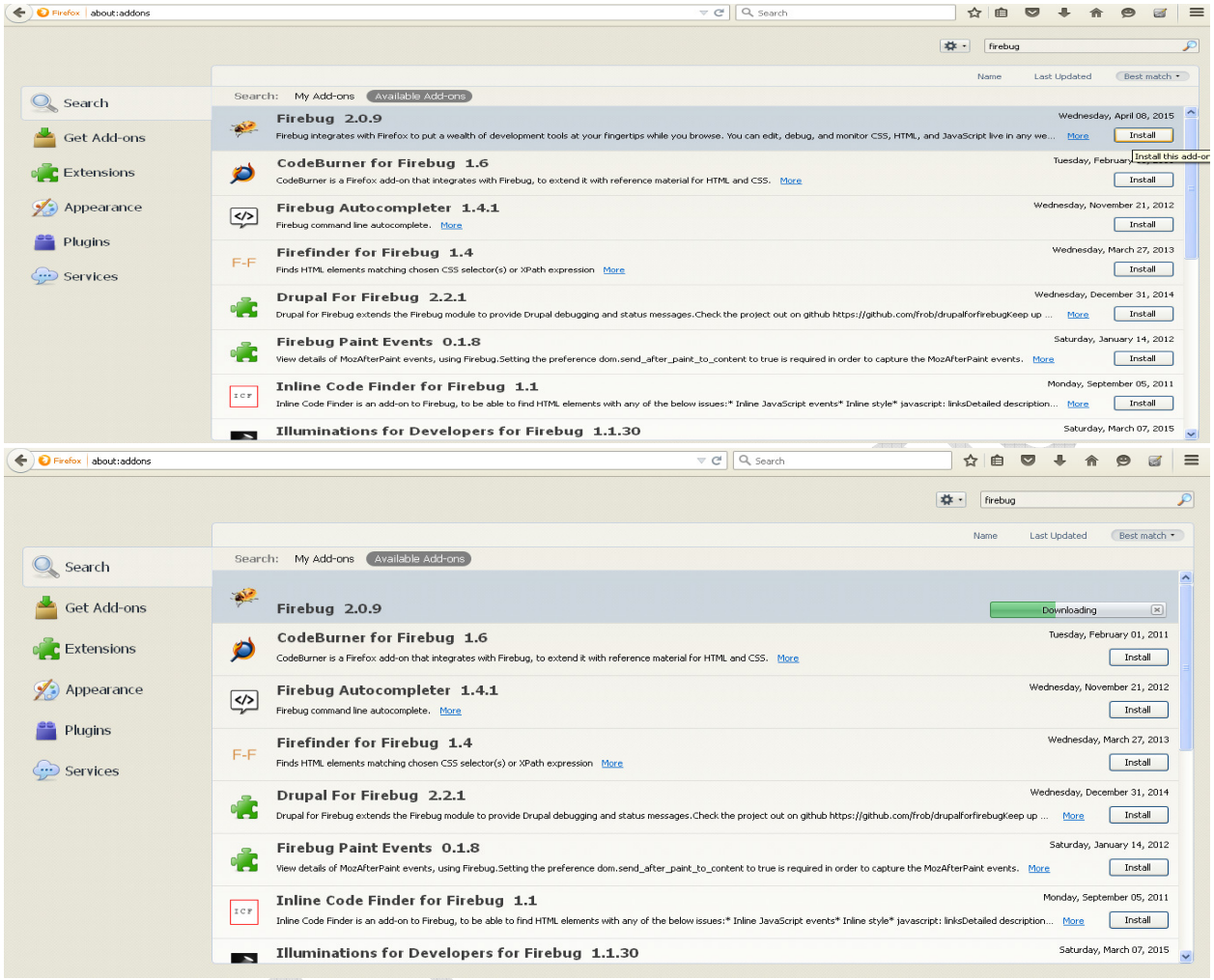
The elements can be found uniquely based on their locator types which we would be discussing later...

How to Install Firebug?

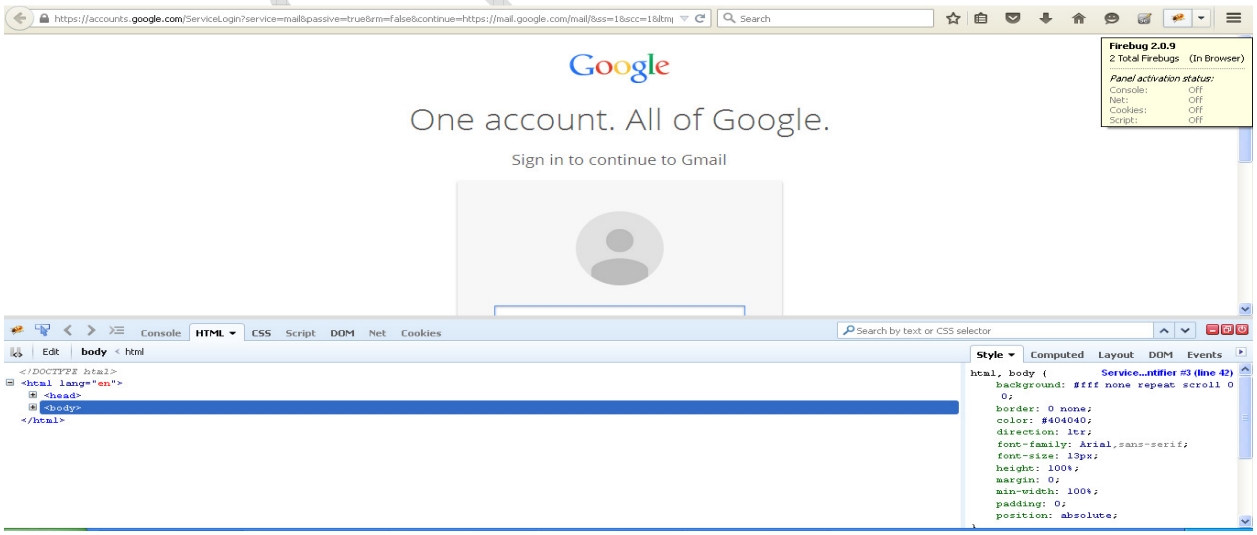
For the ease of understanding, we would bifurcate the installation process into the following steps.

Step -1: Launch the Mozilla Firefox browser and navigate to this Firebug add-on download page. The URL takes us to Firefox add-ons section.

Step -2: Click on the “Add to Firefox” button present on the webpage. Refer the following figure for the same.



Script with firebug:



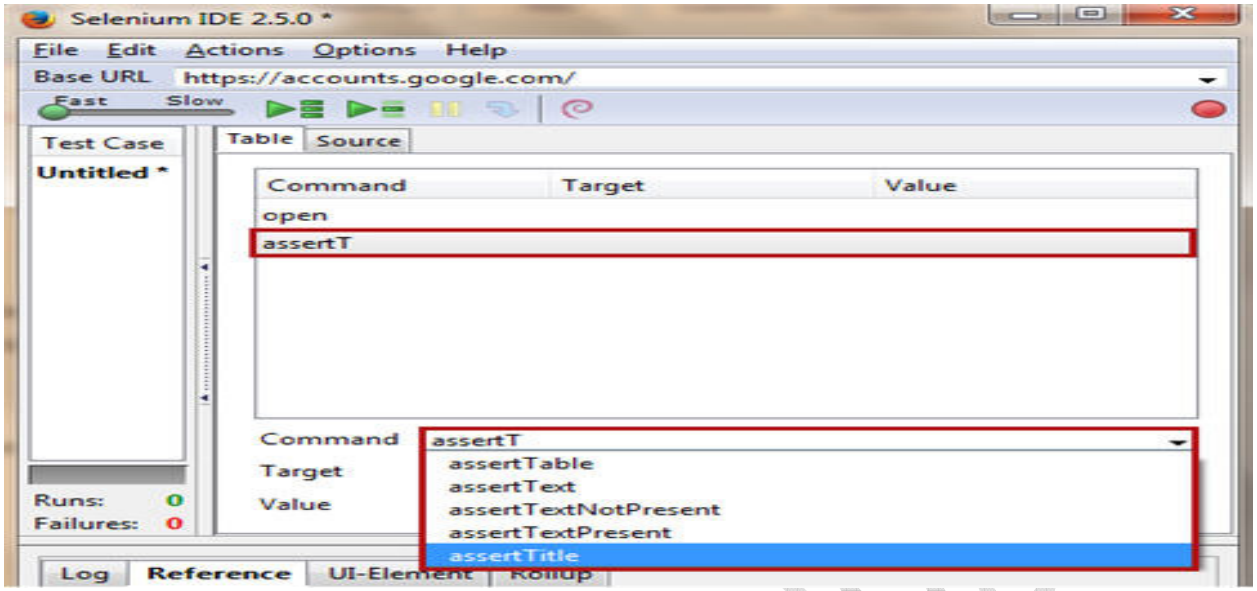
Creating Selenium Script using Firebug:

- Unlike Selenium IDE, In Firebug, we create automated test scripts manually by adding multiple test steps to form a logical and consistent test script.
- Let us follow a progressive approach and understand the process step by step.
- **Scenario:**
- Open “https://accounts.google.com”.
- Assert Title of the application
- Enter an invalid username and invalid password and submit the details to login.
    - **Step 1** – Launch the Firefox and open Selenium IDE from the menu bar.
- Recommendation: While typing commands in the command text box, user can leverage the feature of auto selection. Thus, as soon as the user types a sequence of characters, the matching suggestions would be auto populated.
- User can also click on the dropdown available within the command text box to look at all the commands provided by Selenium IDE.
    - **Step 8** – Now, motion towards the Firebug section within the web browser, expand “head” section of the HTML code. Notice the HTML tag <title>. Thus to assert the title of the webpage, we would require the value of the <title> tag.

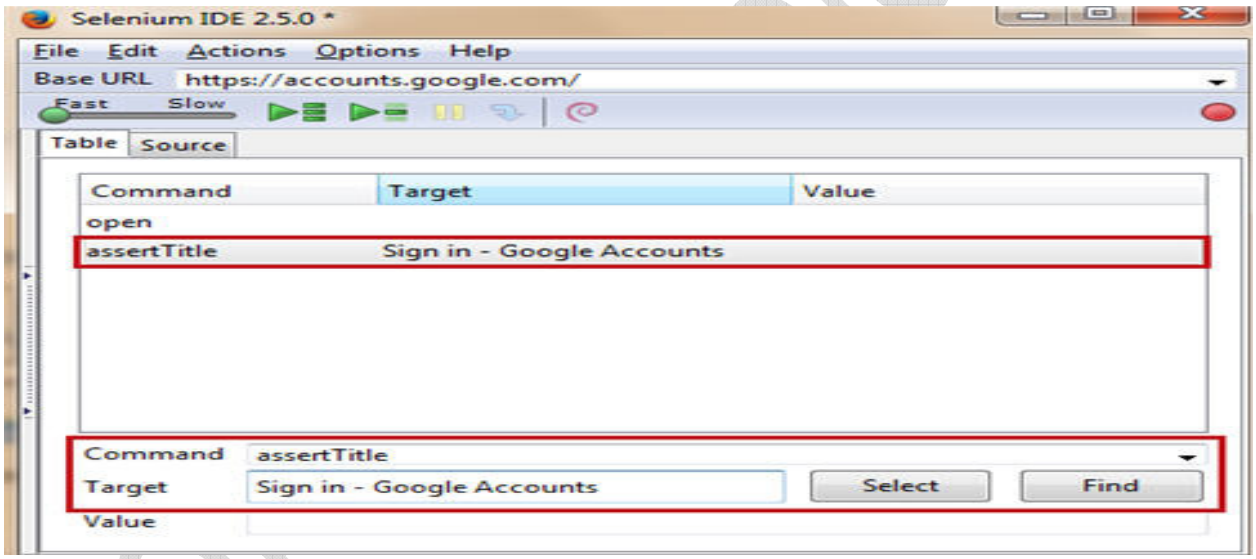
Copy the title of the webpage which is “Sign in – Google Accounts” in our case.

Step 9 – Select the second empty test step within the Editor.

Step 10 – Type “assertTitle” in the command text box present in the Editor Pane. The “assertTitle” command returns the current page title and compares it with the specified title.

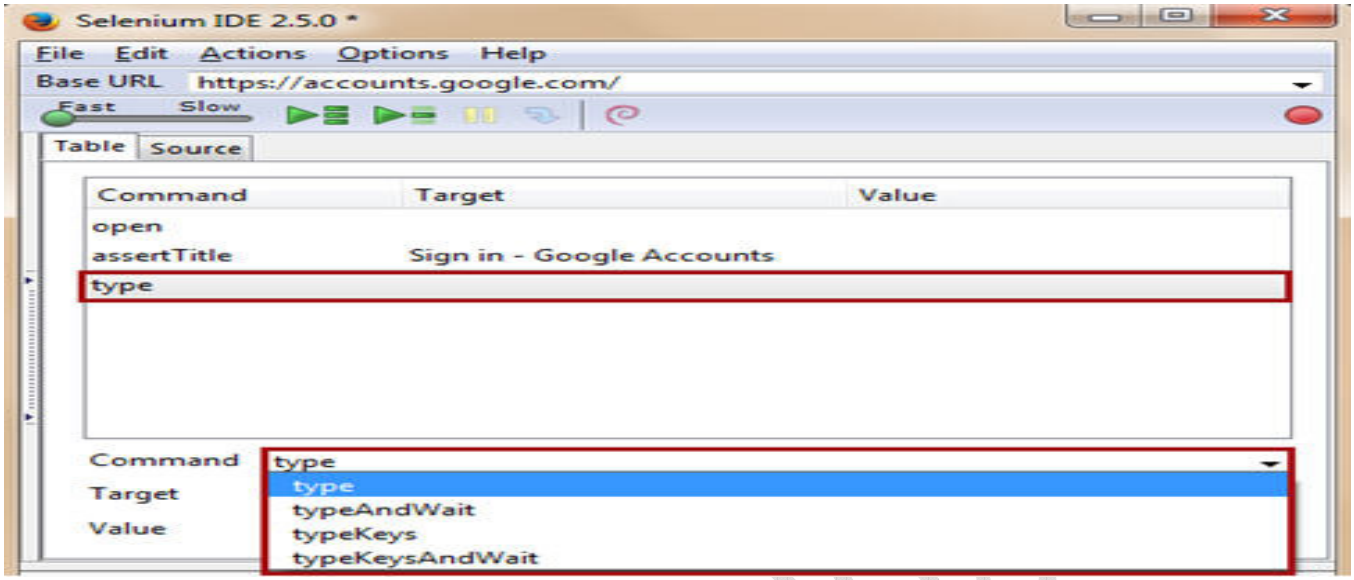


Step 11 – Paste the title copied in step 8 into the Target field of the second.

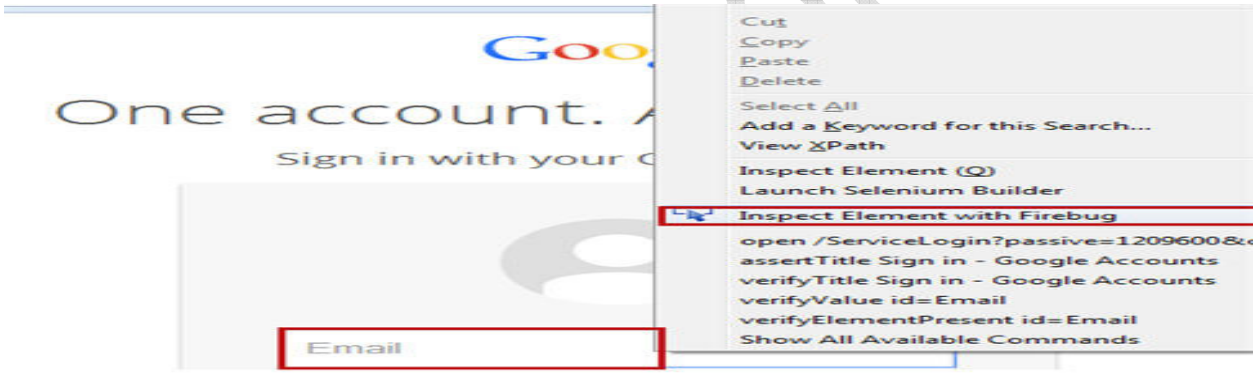


Step 12 – Now select the third empty test step in the Editor Pane

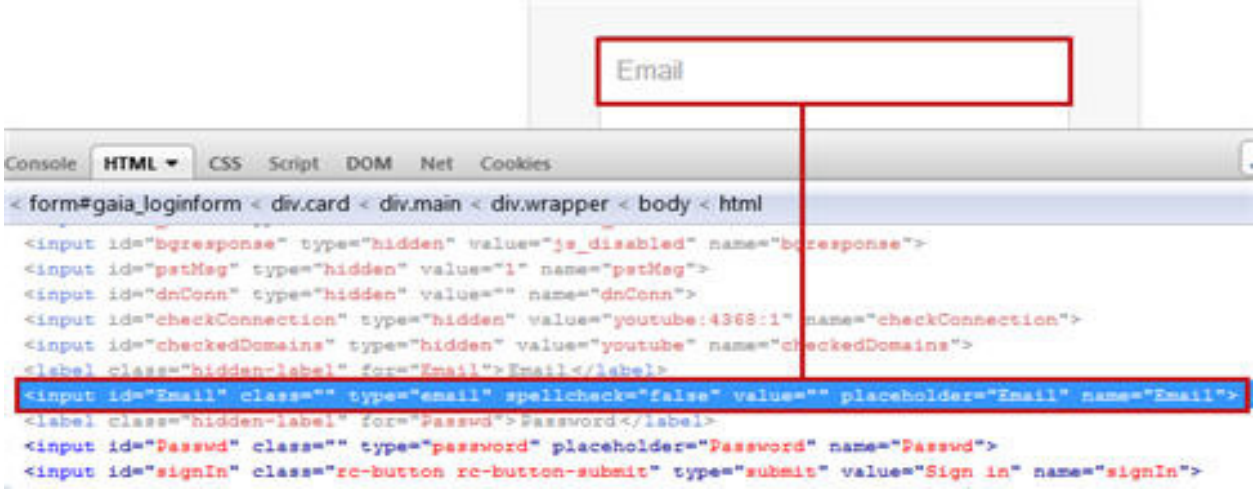
Step 13 – Type "type" command within the command text box. The "type" command enters a value in the specified web element on to the GUI.



Step 14 – Now switch to the web browser, bring the mouse cursor to the "Email" textbox within the login form and press a right click.

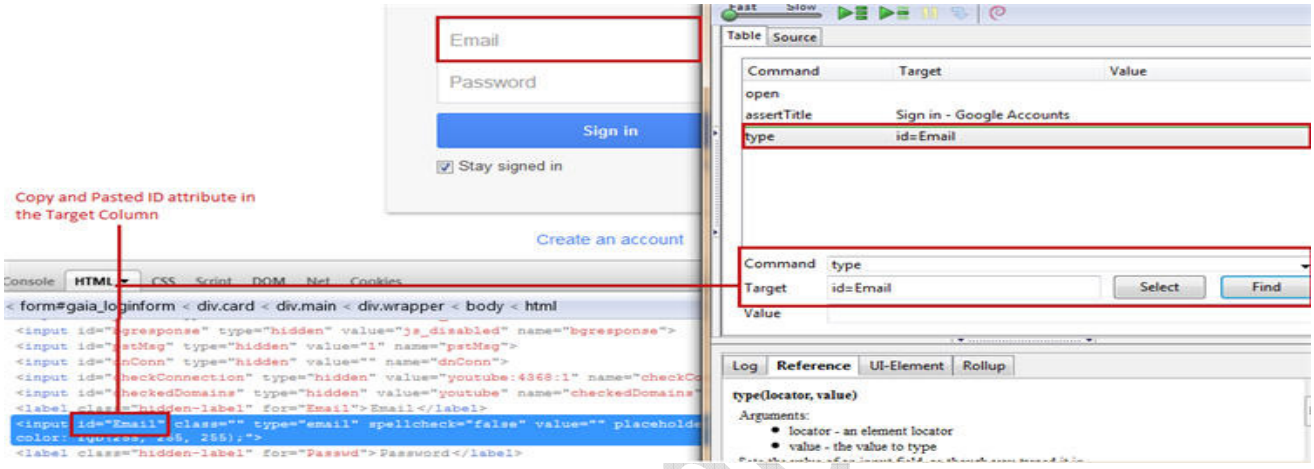


Choose "Inspect Element with Firebug"option. Notice that the Firebug automatically highlights the corresponding HTML code for the web element i.e. "Email Textbox".



Step 15 – The HTML code in the above illustration manifests the distinct property attributes belonging to the "Email" text box. Notice that there are four properties (ID, type, placeholder and name) that uniquely identify the web element on the web page. Thus it's up to the user to choose one or more than one property to identify the web element.

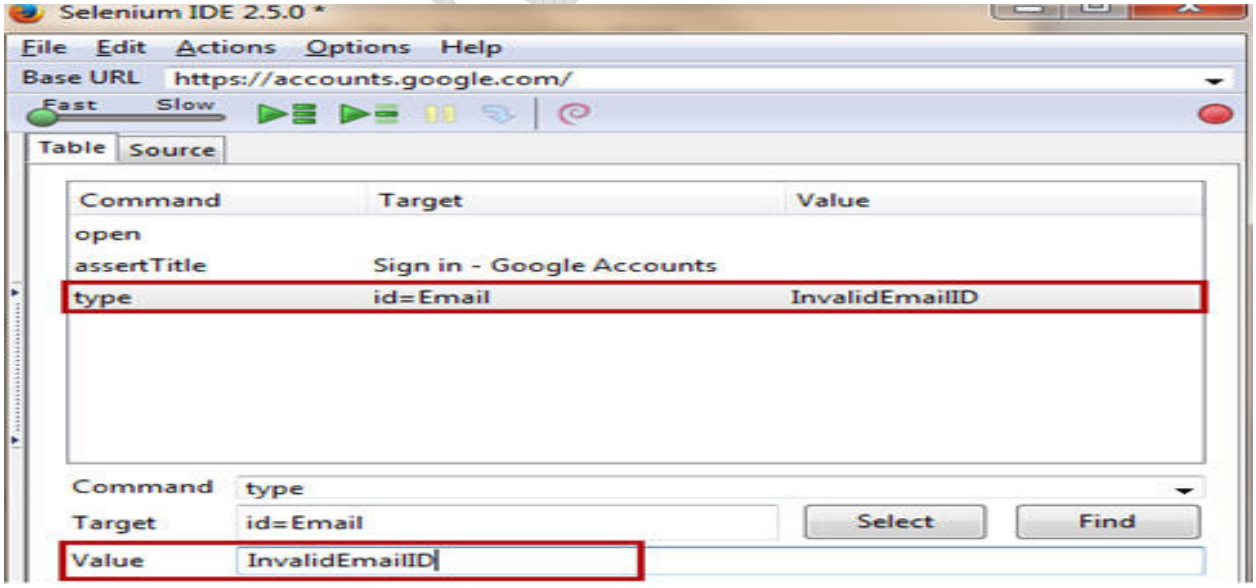
Thus, in this case, we choose ID as the locator. Copy the ID value and paste it in the Target field of the third test step prefixed with "id=" to indicate Selenium IDE to locate a web element having ID as "Email".



Make a note that Selenium IDE is case sensitive, thus type the attribute value carefully and precisely the same as it is displayed in the HTML code.

Step 16 – Click on the Find button to verify if the locator selected finds and locates the designated UI element on the web page.

Step 17 – Now, the next step is to enter the test data in the Value textbox of the third test step within the Editor Pane. Enter "InvalidEmailID" in the Value textbox. User can alter the test data as and when it is desired.

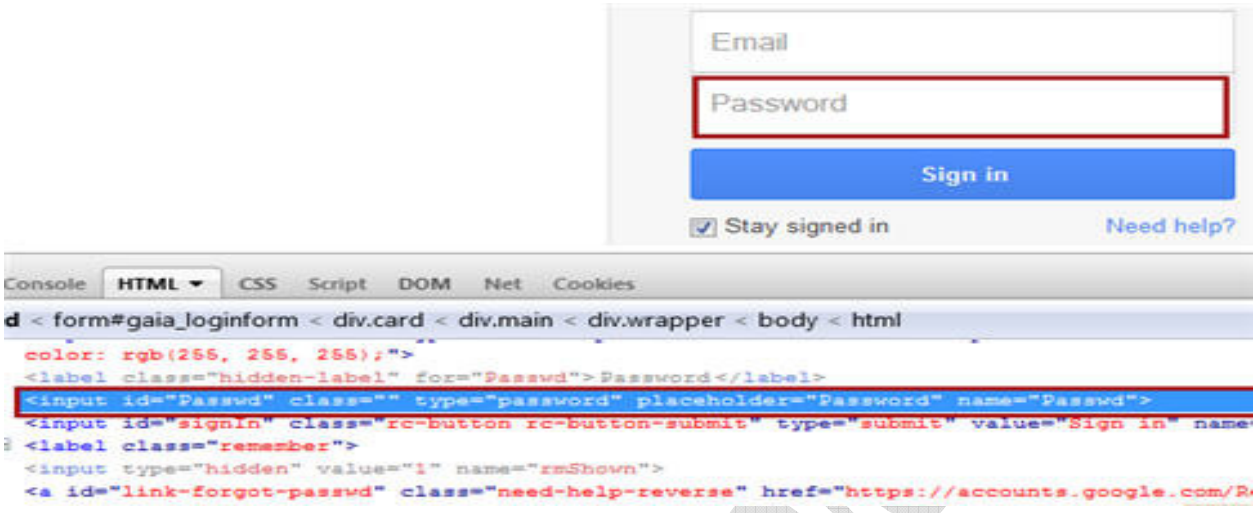


Step 18 – Now select the fourth empty test step in the Editor Pane

Step 19 – Type "type" command within the command text box.

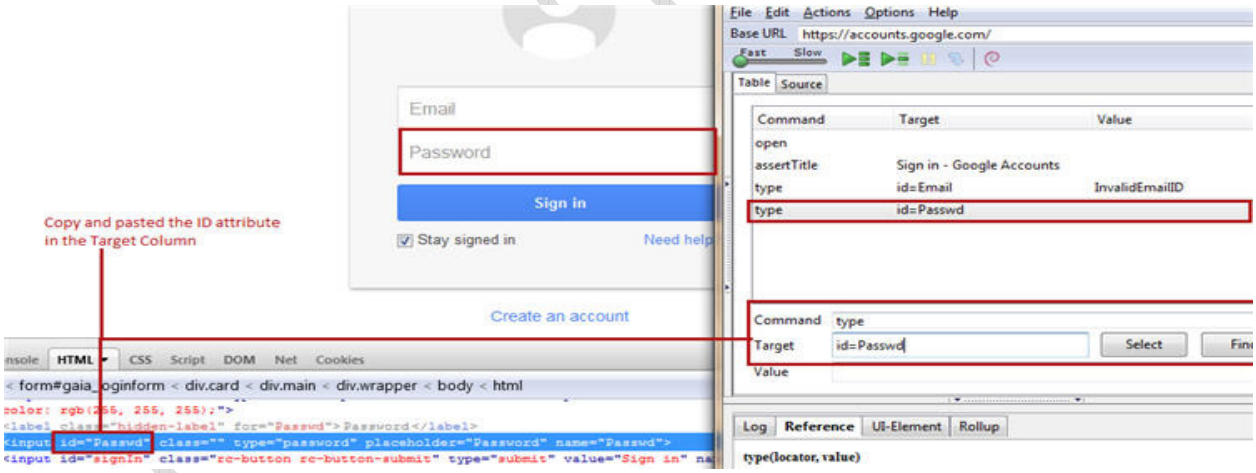
Step 20 – Now switch to the web browser, bring the mouse cursor to the “Password” textbox within the login form and press a right click.

Choose “Inspect Element with Firebug”option.



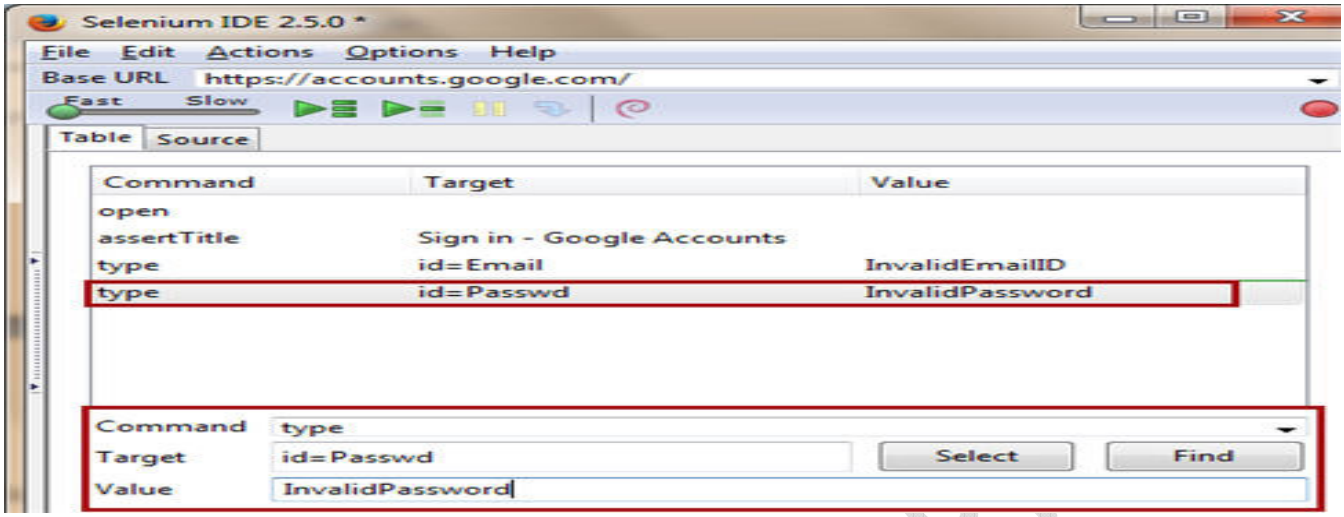
Step 21 – The HTML code below manifests the distinct property attributes belonging to the “Password” text box. Notice that there are four properties (ID, type, placeholder and name) that uniquely identify the web element on the web page. Thus it’s up to the user to choose one or more than one property to identify the web element.

Thus, in this case, we choose ID as the locator. Copy the ID value and paste it in the Target field of the third test step prefixed with “id=”.



Step 22 – Click on the Find button to verify if the locator tabbed finds and locates the designated UI element on the web page.

Step 23 – Now, the next step is to enter the test data in the Value textbox of the fourth test step within the Editor Pane. Enter “InvalidPassword” in the Value textbox. User can alter the test data as and when it is desired.

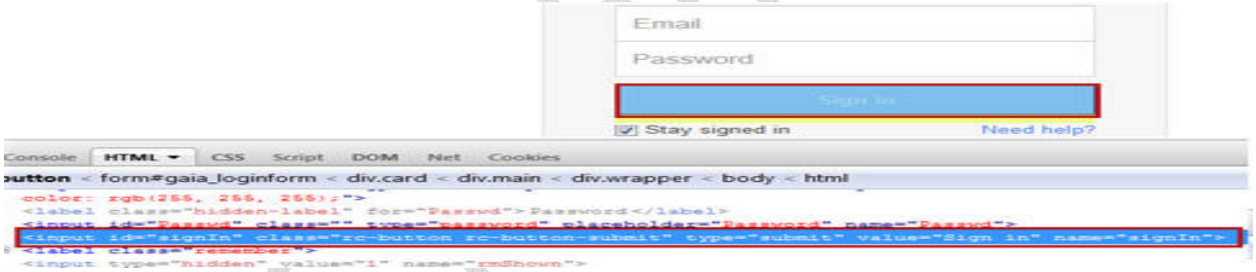


Step 24 – Now select the fifth empty test step in the Editor Pane

Step 25 – Type "click" command within the command text box. The "click" command clicks on a specified web element within the web page.

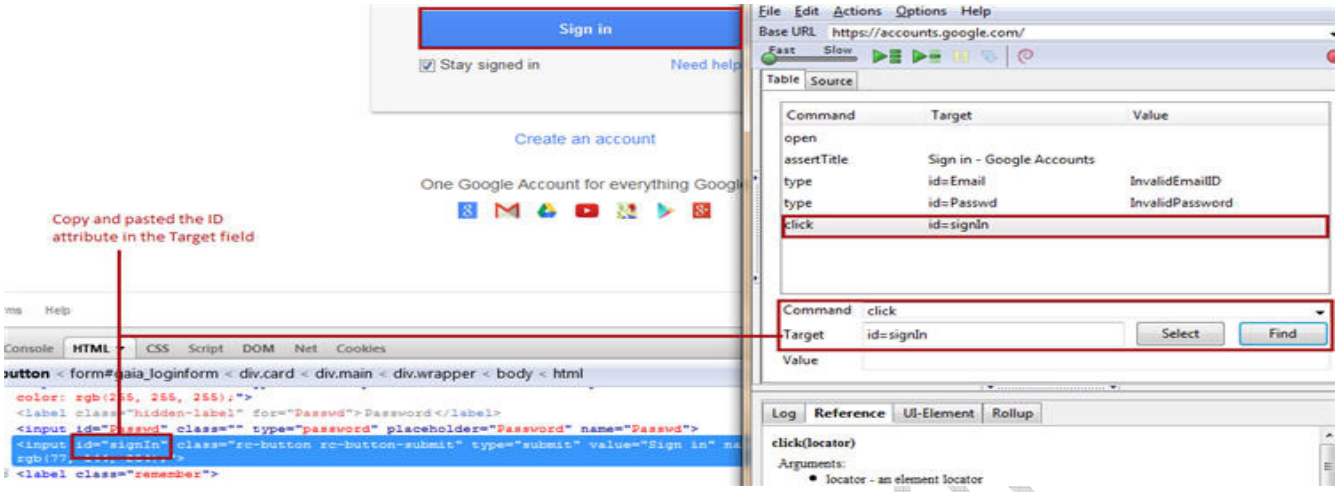
Step 26 – Now switch to the web browser, bring the mouse cursor to the "Sign in" button within the login form and press a right click.

Choose "Inspect Element with Firebug"option.



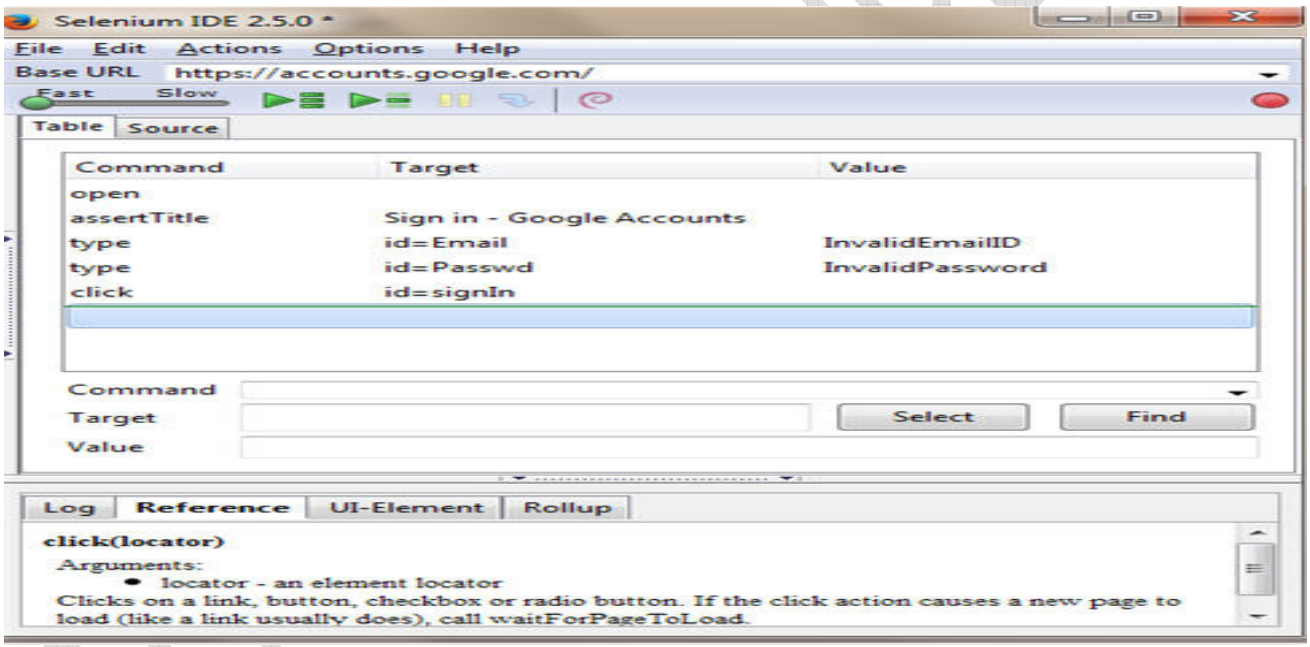
Step 27 – The HTML code below manifests the distinct property attributes belonging to the "Sign in" button.

Choose ID as the locator. Copy the ID value and paste it in the Target field of the third test step prefixed with "id=".



Step 28 – Click on the Find button to verify if the locator picked finds and locates the designated UI element on the web page.

The test script is completed now. Refer the following illustration to view the finished test script.



Step 29 – Play back the created test script and Save it in the same way we did in the previously.

How to Identify Web Elements Using Selenium Xpath and Other Locators

- **What is Locator?**

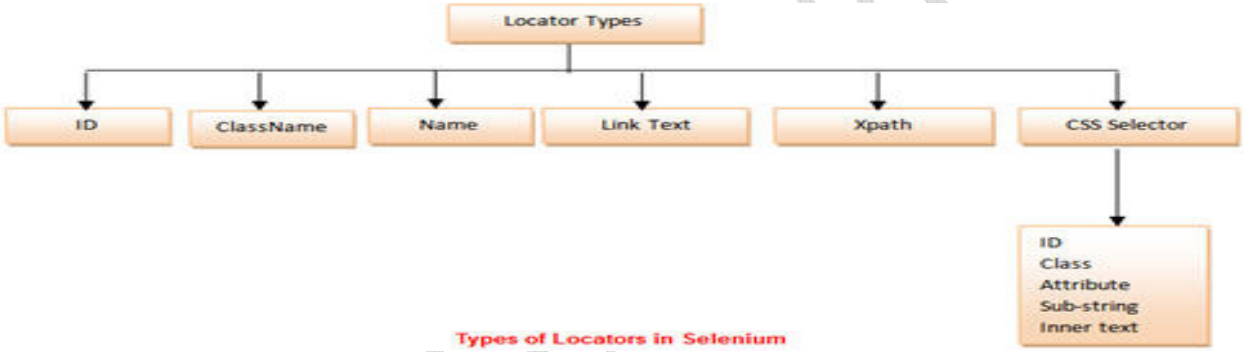
Locator can be termed as an address that identifies a web element uniquely within the webpage. Locators are the HTML properties of a web element which tells the Selenium about the web element it need to perform action on.

- **There is a diverse range of web elements. The most common amongst them are:**
  - Text box

- Button
- Drop Down
- Hyperlink
- Check Box
- Radio Button

- **Types of Locators:**
- Identifying these elements has always been a very tricky subject and thus it requires an accurate and effective approach.
- Thereby, we can assert that more effective the locator, more stable will be the automation script.
- Essentially every Selenium command requires locators to find the web elements.

Thus, to identify these web elements accurately and precisely we have different types of locators



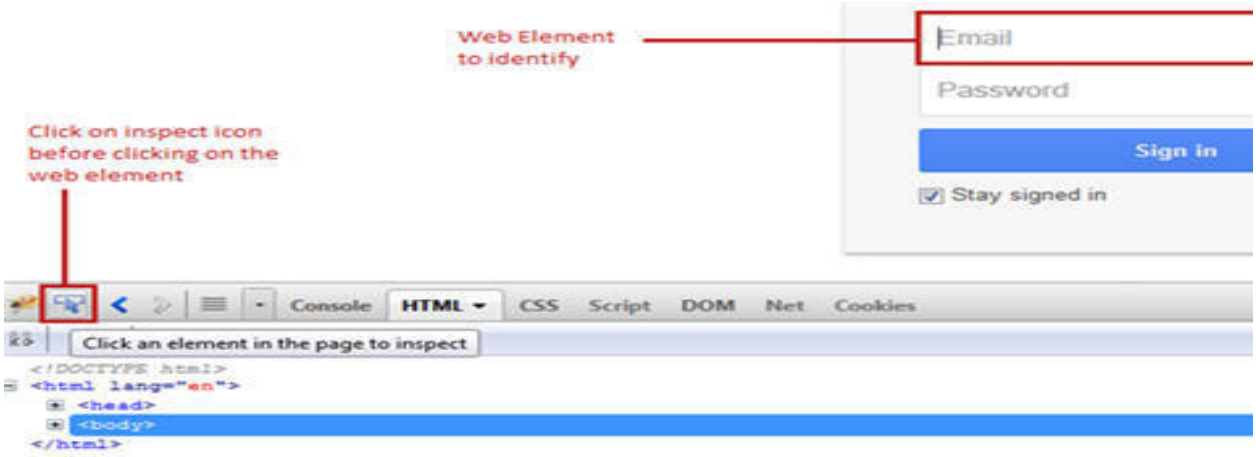
Types of Locators in Selenium

- **Using ID as a Locator**

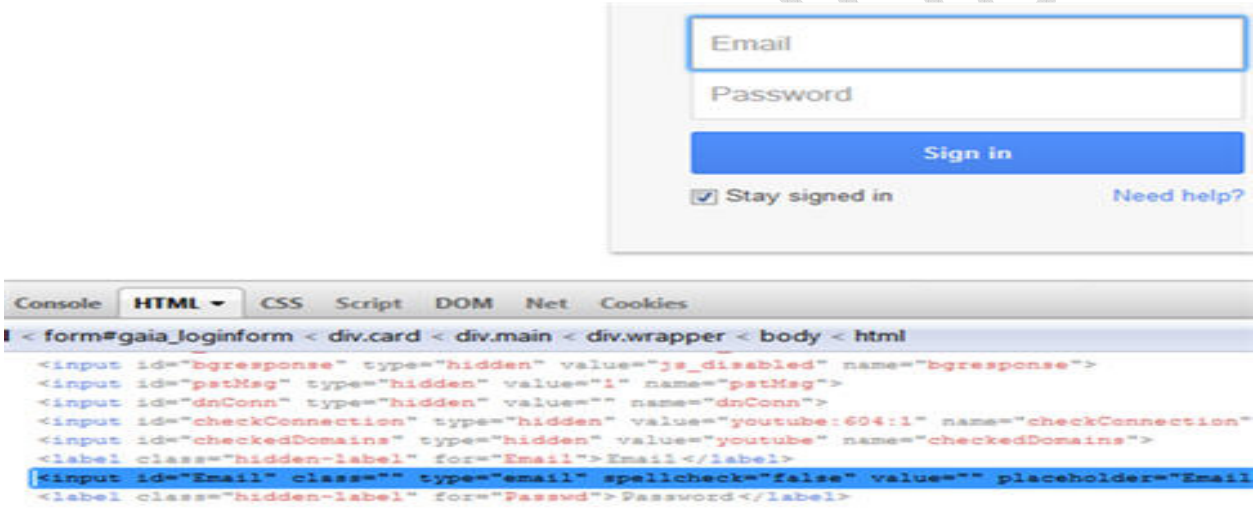
The best and the most popular method to identify web element is to use ID. The ID of an each element is alleged to be unique.

- **<u>Finding an ID of a web element using Firebug</u>**

Step 1: Launch the web browser (Firefox) and navigate to “https://accounts.google.com/”.
Step 2: Open firebug (either by pressing F12 or via tools).
Step 3: Click on the inspect icon to identify the web element.



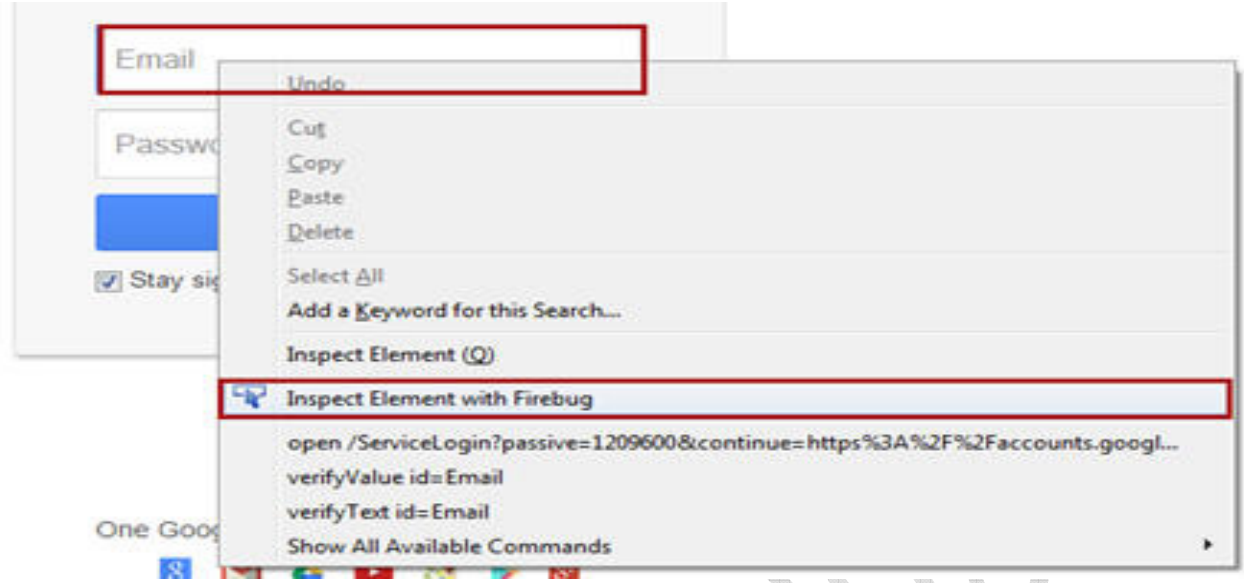
Step 4: Hover on the web element (Email textbox in our case) on which we desire to perform some action. In the firebug section, one can see the corresponding html tags being highlighted.



Step 5: Be cognizant about the ID attribute and take a note of it. Now we need to verify if the ID indentified is able to find the element uniquely and flawlessly.

- **Syntax**: id = id of the element
- In our case, the id is “Email”.
- Alternative approach:

Instead of following step 2 to 4, we can directly locate / inspect the web element by right clicking on the web element (Email Textbox) whose locator value we need to inspect and clicking on the option “Inspect Element with Firebug”. Thus, this click event triggers the expansion of firebug section and the corresponding html tag would be highlighted.



Verify the locator value

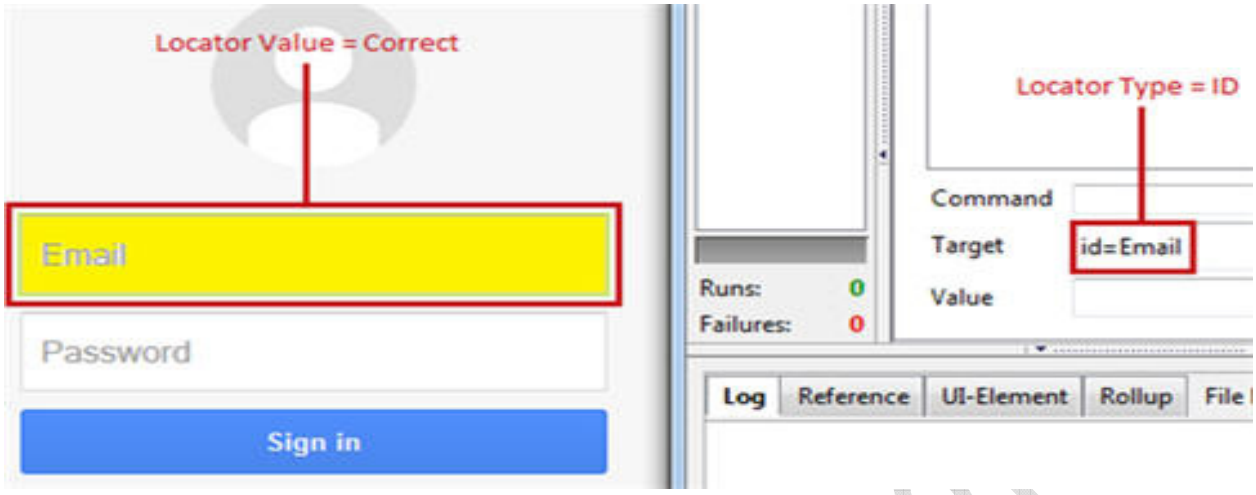
Assuming that the browser is open and is re-directed to "https://accounts.google.com/".

- **Step 1**: Launch Selenium IDE.
- **Step 2**: Click on the first row in the editor section.
- **Step 3**: Type "id=Email" i.e. the locator value in the target box.
- **Step 4**: Click on the Find Button. If the provided locator value is legitimate then the Email textbox will be highlighted with yellow color with a florescent green border around the field. If the locator value provided is incorrect, an error message would be printed in the log pane at the bottom of Selenium IDE.
- **Case 1** – Locator Value = Correct
- **Step 5**: In order to verify further, user can also execute "type" command against the given target by providing some value in the "Value" field. If the execution of the command enters the specified value in the Email text box that means the identified locator type is correct and accessible.

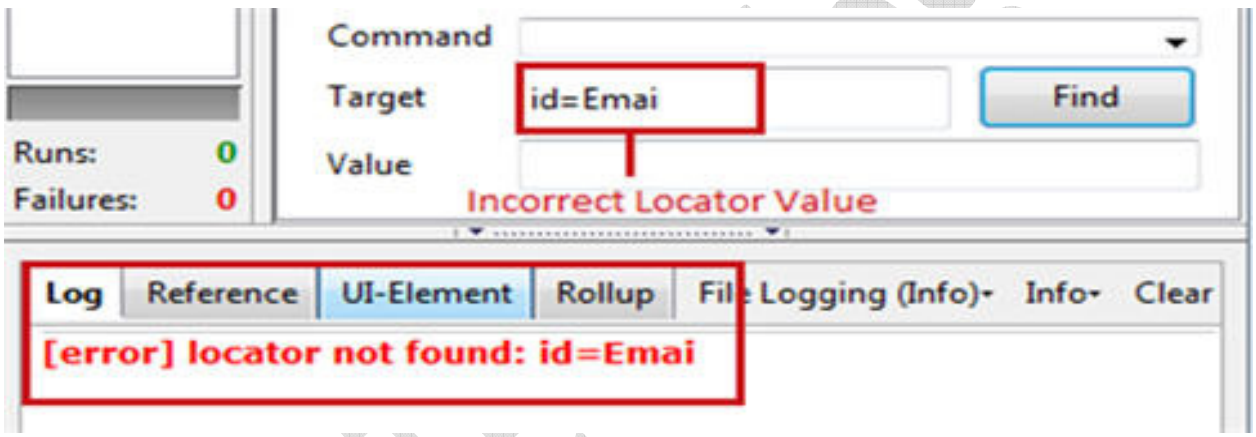
- **Using ClassName as a Locator**
- There is only a subtle difference between using ID as a locator and using classname as a locator.
- In this sample, we would access "Need Help?" hyperlink present at the bottom of the login form at gmail.com.
- **Finding a classname of a web element using Firebug**

Step 1: Locate / inspect the web element ("Need help?" link in our case) by right clicking on the web element whose locator value we need to inspect and clicking on the option "Inspect Element with Firebug".

Step 2: Be cognizant about the classname attribute and take a note of it. Now we need to verify if the classname indentified is able to find the element uniquely and accurately.



Case 2 – Locator Value = Incorrect .



Step 5: In order to verify further, user can also execute "type" command against the given target by providing some value in the "Value" field. If the execution of the command enters the specified value in the Email text box that means the identified locator type is correct and accessible.

Using ClassName as a Locator

There is only a subtle difference between using ID as a locator and using classname as a locator.
In this sample, we would access "Need Help?" hyperlink present at the bottom of the login form at gmail.com.

Finding a classname of a web element using Firebug

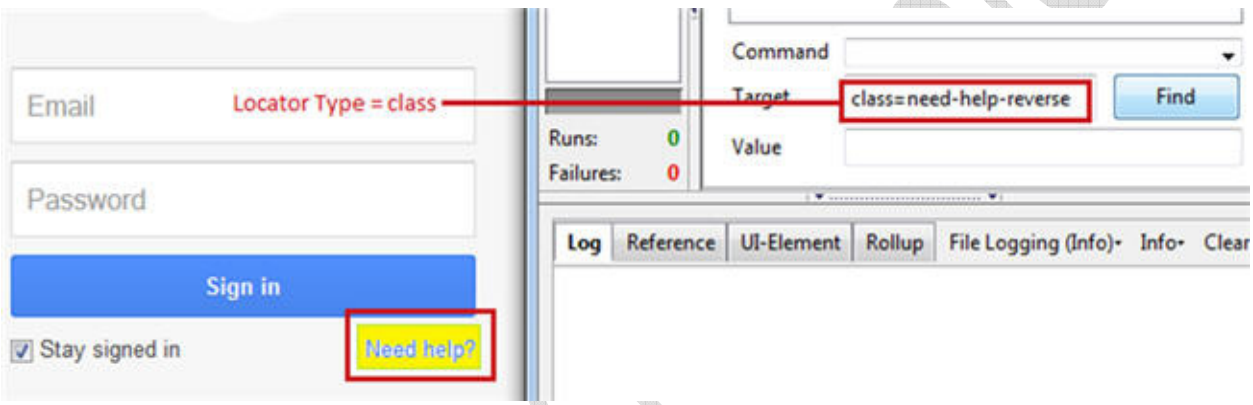
Step 1: Locate / inspect the web element ("Need help?" link in our case) by right clicking on the web element whose locator value we need to inspect and clicking on the option "Inspect Element with Firebug".

Step 2: Be cognizant about the classname attribute and take a note of it. Now we need to verify if the classname indentified is able to find the element uniquely and accurately.

```
<label class="remember">
<input type="hidden" value="1" name="rmShown">
<a id="link-forgot-passwd" class="need-help-reverse" href="https://account
continue=http%3A%2F%2Fmail.google.com%2Fmail%2F"> Need help? </a>
</form>
```

- **Syntax:** class = classname of the element
- In our case, the classname is "need-help-reverse"
- **Verify the locator value**

Step 1: Type "class= need-help-reverse" in the target box in the Selenium IDE.
Step 2: Click on the Find Button. Notice that the hyperlink will be highlighted with yellow color with a florescent green border around the field.



Using name as a Locator
Locating a web element using name is very much analogous to previous two locator types. The only difference lies in the syntax. In this sample, we would access "Password" text box present in the login form at gmail.com. **Syntax:** name = name of the element
In our case, the name is "Passwd".
Verify the locator value
Step 1: Type "name= Passwd" in the target box and click on the Find Button. Notice that the "Password" textbox would be highlighted.
Using Link Text as a Locator
All the hyperlinks on a web page can be indentified using Link Text. The links on a web page can be determined with the help of anchor tag (<a>). The anchor tag is used to create the hyperlinks on a web page and the text between opening and closing of anchor tags constitutes the link text (<a>Some Text).
In this sample, we would access "Create an account" link present at the bottom of the login form at gmail.com.
Finding a link text of a web element using Firebug
Step 1: Locate / inspect the web element ("Create an account" link in our case) by right clicking on the web element whose locator value we need to inspect and clicking on the option "Inspect Element with Firebug".

Key Points:

The success rate of finding an element using Xpath is too high. Along with the previous statement, Xpath can find relatively all the elements within a web page. Thus, Xpaths can be used to locate elements having no id, class or name.

Creating a valid Xpath is a tricky and complex process. There are plug-ins available to generate Xpath but most of the times, the generated Xpaths fails to identify the web element correctly.

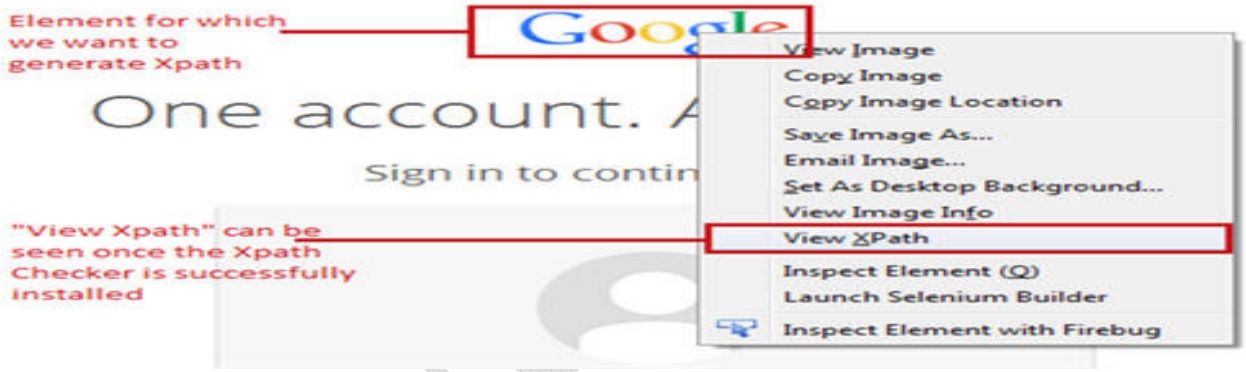
While creating xpath, user should be aware of the various nomenclatures and protocols.

Selenium Xpath Examples

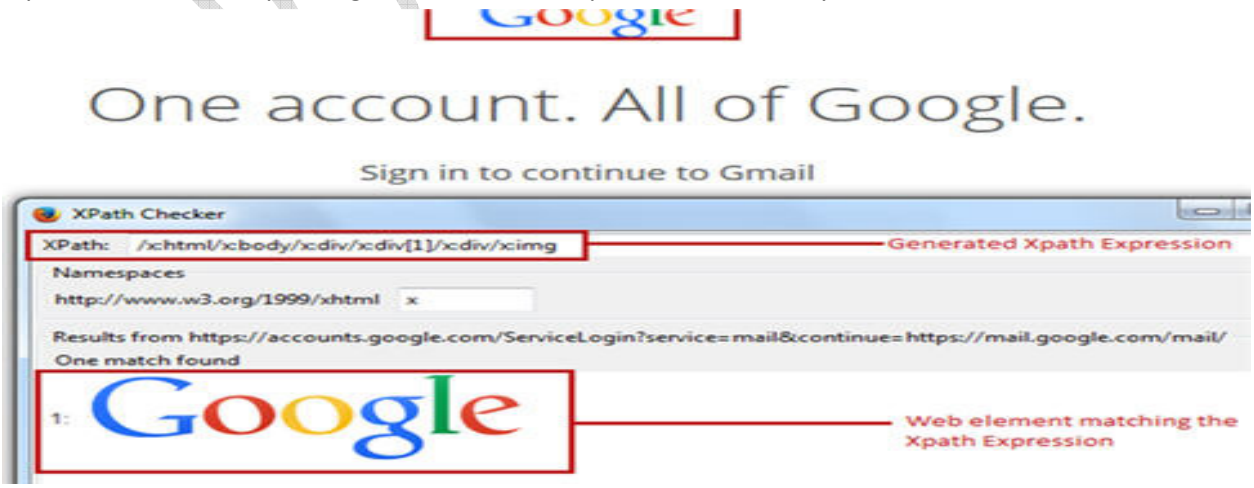
Xpath Checker

Creating Xpath becomes a little simpler by using Xpath Checker. Xpath Checker is a firefox add-on to automatically generate Xpath for a web element. The add-on can be downloaded and installed like any other plug-in. The plug-in can be downloaded from "https://addons.mozilla.org/en-US/firefox/addon/xpath-checker/".

As soon as the plug-in is installed, it can be seen in the context menu by right clicking any element whose xpath we want to generate.



Click on the "View Xpath" to see the Xpath expression of the element. An editor window would appear with the generated Xpath expression. Now user has the liberty to edit and modify the generated Xpath expression. The corresponding results would be updated cumulatively.



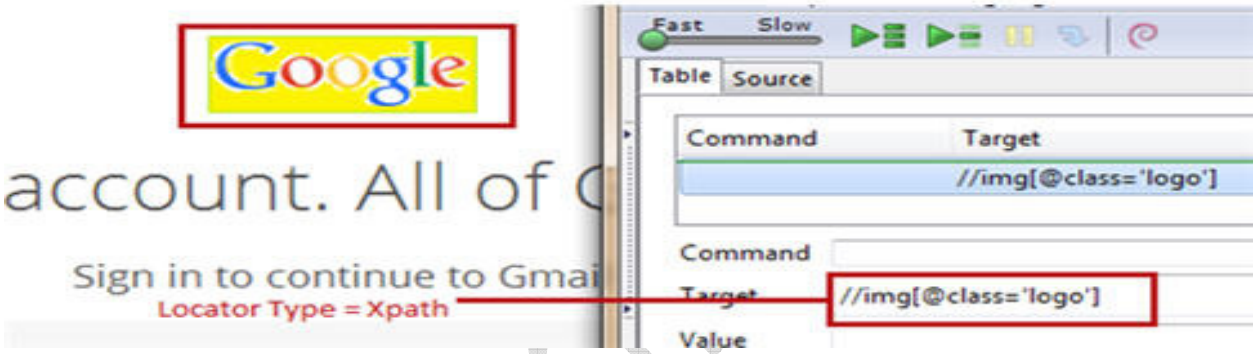
- Note that the Xpath Checker is available for other browsers as well.

- But re-iterating the fact, that most of the times, the generated Xpaths fails to identify the web element rightly. Thus, it is recommended to create our own Xpath following the pre defined rules and protocols.
- In this sample, we would access “Google” image present at the top of the login form at gmail.com.
- **Creating a Xpath of a web element**

Step 1: Type “//img[@class=’logo’]” i.e. the locator value in the target box within the Selenium IDE.

Syntax: Xpath of the element

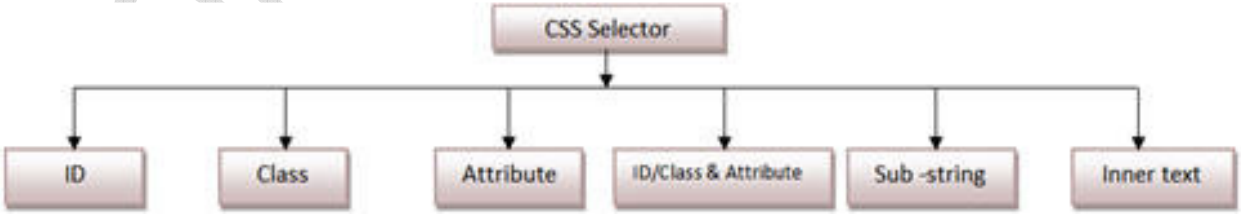
Step 2: Click on the Find Button. Notice that the image would be highlighted with yellow color with a florescent green border around the field.



Using CSS Selector as a Locator:

CSS Selector is combination of an element selector and a selector value which identifies the web element within a web page. The composite of element selector and selector value is known as Selector Pattern.

Selector Pattern is constructed using HTML tags, attributes and their values. The central theme behind the procedure to create CSS Selector and Xpath are very much similar underlying the only difference in their construction protocol. Like Xpath, CSS selector can also locate web elements having no ID, class or Name. So now gearing ahead, let us discuss the primitive types of CSS Selectors:



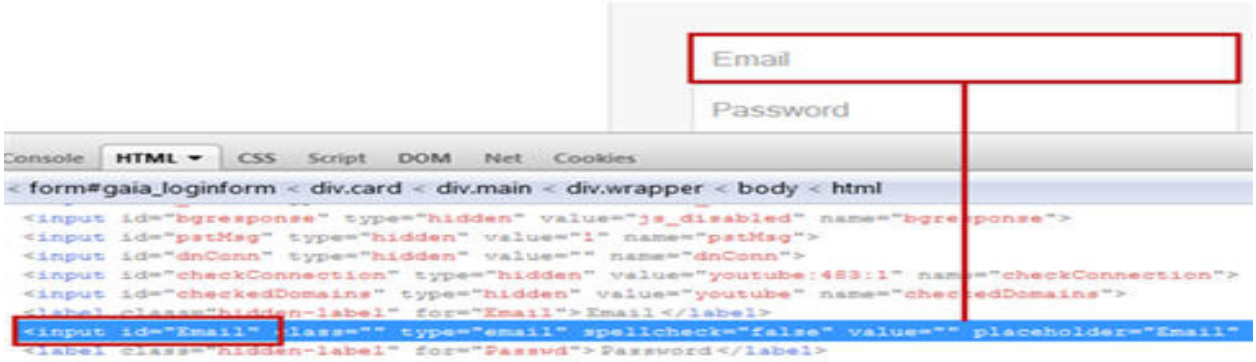
CSS Selector: ID

In this sample, we would access “Email” text box present in the login form at Gmail.com.

The Email textbox has an ID attribute whose value is defined as “Email”. Thus ID attribute and its value can be used to create CSS Selector to access the email textbox.

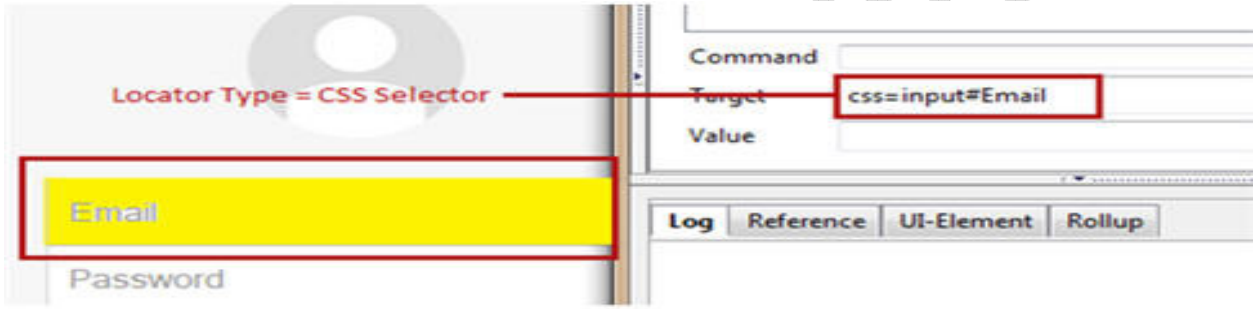
Creating CSS Selector for web element

Step 1: Locate / inspect the web element ("Email" textbox in our case) and notice that the html tag is "input" and value of ID attribute is "Email" and both of them collectively make a reference to the "Email Text box". Hence the above data would be used to create CSS Selector.



- **Verify the locator value**

Step 1: Type "css=input#Email" i.e. the locator value in the target box in the Selenium IDE and click on the Find button. Notice that the Email Text box would be highlighted.



Syntax: css=<HTML tag><#><Value of ID attribute>

HTML tag – It is tag which is used to denote the web element which we want to access.

– The hash sign is used to symbolize ID attribute. It is mandatory to use hash sign if ID attribute is being used to create CSS Selector.

Value of ID attribute – It is the value of an ID attribute which is being accessed. The value of ID is always preceded by a hash sign.

While specifying CSS Selector in the target text box of Selenium IDE, always remember to prefix it with "css=". The sequence of the above artifacts is inalterable. If two or more web elements have the same HTML tag and attribute value, the first element marked in the page source will be identified.

CSS Selector: Class

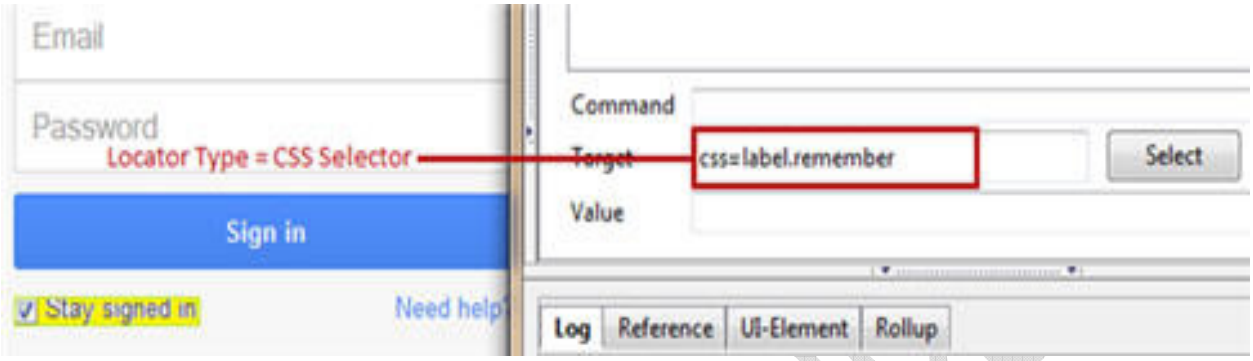
In this sample, we would access "Stay signed in" check box present below the login form at gmail.com. The "Stay signed in" check box has a Class attribute whose value is defined as "remember". Thus Class attribute and its value can be used to create CSS Selector to access the designated web element. Locating an element using Class as a CSS Selector is very much similar to using ID, the lone difference lies in their syntax formation.

Creating CSS Selector for web element

Step 1: Locate / inspect the web element ("Stay signed in" check box in our case) and notice that the html tag is "label" and value of ID attribute is "remember" and both of them collectively make a reference to the "Stay signed in check box".

Verify the locator value

Step 1: Type "css=label.remember" i.e. the locator value in the target box in the Selenium IDE and click on the Find Button. Notice that the "Stay signed in" check box would be highlighted.



Syntax

css=<HTML tag><.><Value of Class attribute>

. – The dot sign is used to symbolize Class attribute. It is mandatory to use dot sign if Class attribute is being used to create CSS Selector. The value of Class is always preceded by a dot sign.

CSS Selector: Attribute

In this sample, we would access "Sign in" button present below the login form at gmail.com.

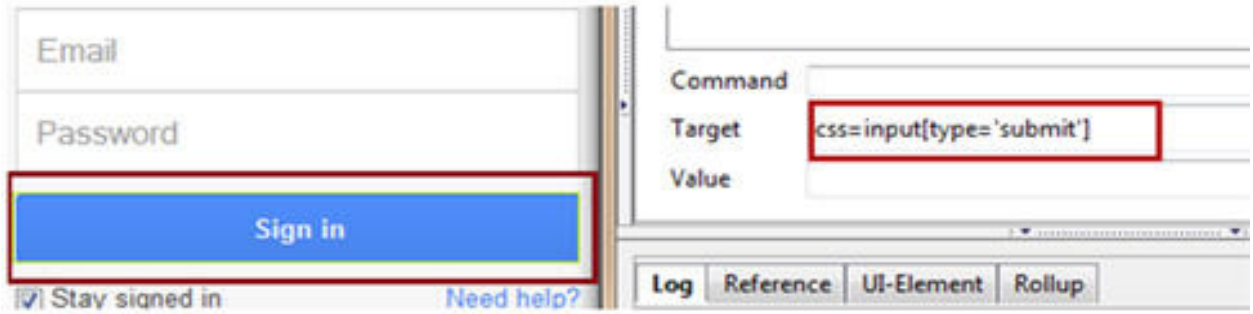
The "Sign in" button has a type attribute whose value is defined as "submit". Thus type attribute and its value can be used to create CSS Selector to access the designated web element.

Creating CSS Selector for web element

Step 1: Locate / inspect the web element ("Sign in" button in our case) and notice that the html tag is "input", attribute is type and value of type attribute is "submit" and all of them together make a reference to the "Sign in" button

Verify the locator value

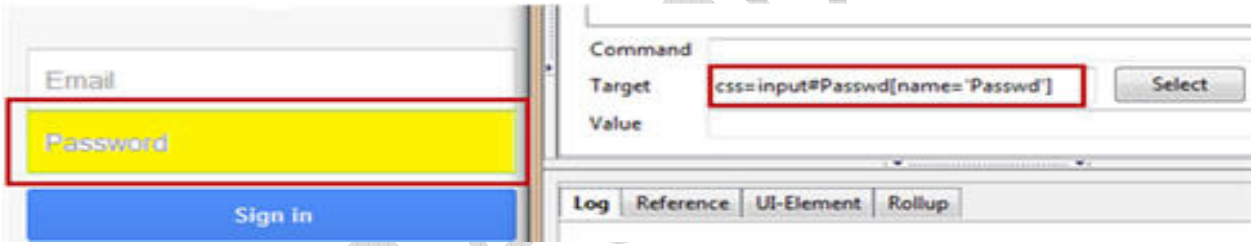
Step 1: Type "css=input[type='submit']" i.e. the locator value in the target box in the Selenium IDE and click on the Find Button. Notice that the "Sign in" button would be highlighted.



Syntax

css=<HTML tag><[attribute=Value of attribute]>

Attribute – It is the attribute we want to use to create CSS Selector. It can value, type, name etc. It is recommended to choose an attribute whose value uniquely identifies the web element.
Value of attribute – It is the value of an attribute which is being accessed.
CSS Selector: ID/Class and attribute
In this sample, we would access "Password" text box present in the login form at gmail.com.
The "Password" text box has an ID attribute whose value is defined as "Passwd", type attribute whose value is defined as "password". Thus ID attribute, type attribute and their values can be used to create CSS Selector to access the designated web element.
Creating CSS Selector for web element
Step 1: Locate / inspect the web element ("Password" text box in our case) and notice that the html tag is "input", attributes are ID and type and their corresponding values are "Passwd" and "password" and all of them together make a reference to the "Password" textbox.
Verify the locator value
Step 1: Type "css=input#Passwd[name='Passwd']" i.e. the locator value in the target box in the Selenium IDE and click on the Find Button. Notice that the "Password" text box would be highlighted.



Syntax
css=<HTML tag><. Or #><value of Class or ID attribute><[attribute=Value of attribute]>
Two or more attributes can also be furnished in the syntax. For example, *"css=input#Passwd[type='password'][name='Passwd']".*
CSS Selector: Sub-string
CSS in Selenium allows matching a partial string and thus deriving a very interesting feature to create CSS Selectors using sub strings. There are three ways in which CSS Selectors can be created based on mechanism used to match the sub string.
Types of mechanisms
All the underneath mechanisms have symbolic significance.

- Match a prefix
- Match a suffix
- Match a sub string

Match a prefix

- It is used to correspond to the string with the help of a matching prefix.

Syntax
css=<HTML tag><[attribute^=prefix of the string]>

^ – Symbolic notation to match a string using prefix.

Prefix – It is the string based on which match operation is performed. The likely string is expected to start with the specified string.

For Example: Let us consider "Password textbox", so the corresponding CSS Selector would be:

css=input#Passwd[name^='Pass']

Match a suffix

It is used to correspond to the string with the help of a matching suffix.

Syntax

css=<HTML tag><[attribute$=suffix of the string]>

 # – Symbolic notation to match a string using suffix.

Suffix – It is the string based on which match operation is performed. The likely string is expected to ends with the specified string.

For Example: Lets again consider "Password textbox", so the corresponding CSS Selector would be:

css=input#Passwd[name$='wd']

Match a sub string

It is used to correspond to the string with the help of a matching sub string.

Syntax

- css=<HTML tag><[attribute*=sub string]>
- *** –** Symbolic notation to match a string using sub string.

  Sub string – It is the string based on which match operation is performed. The likely string is expected to have the specified string pattern.

For Example: Lets again consider "Password textbox", so the corresponding CSS Selector would be:

css=input#Passwd[name$='wd']

- **CSS Selector: Inner text**

Inner text helps us identify and create CSS Selector using a string pattern that the HTML Tag manifests on the web page. Consider, "Need help?" hyperlink present below the login form at gmail.com.

The anchor tag representing the hyperlink has a text enclosed within. Thus this text can be used to create CSS Selector to access the designated web element.

Matching Text Patterns

Like locators, *patterns* are a type of parameter frequently required by Selenese commands.

Examples of commands which require patterns

are **verifyTextPresent**, **verifyTitle**,**verifyAlert**, **assertConfirmation**, **verifyText**, and **verifyPrompt**.

And as has been mentioned above, link locators can utilize a pattern. Patterns allow you to *describe*, via the use of special characters, what text is expected rather than having to specify that text exactly.

There are three types of patterns: *globbing*, *regular expressions*, and *exact*.

Globbing Patterns

Most people are familiar with globbing as it is utilized in filename expansion at a DOS or Unix/Linux command line such as ls *.c. In this case, globbing is used to display all the files ending with a .c extension that exist in the current directory. Globbing is fairly limited. Only two special characters are supported in the Selenium implementation:

***** which translates to "match anything," i.e., nothing, a single character, or many characters.
[ ] (*character class*) which translates to "match any single character found inside the square brackets." A dash (hyphen) can be used as a shorthand to specify a range of characters (which are contiguous in the ASCII character set). A few examples will make the functionality of a character class clear:
[aeiou] matches any lowercase vowel
[0-9] matches any digit
[a-zA-Z0-9] matches any alphanumeric character
In most other contexts, globbing includes a third special character, the **?**. However, Selenium globbing patterns only support the asterisk and character class.
To specify a globbing pattern parameter for a Selenese command, you can prefix the pattern with a **glob:** label. However, because globbing patterns are the default, you can also omit the label and specify just the pattern itself.

- Below is an example of two commands that use globbing patterns. The actual link text on the page being tested was "Film/Television Department"; by using a pattern rather than the exact text, the **click** command will work even if the link text is changed to "Film & Television Department" or "Film and Television Department". The glob pattern's asterisk will match "anything or nothing" between the word "Film" and the word "Television".

Command	Target
click	link=glob:Film*Television Department
verifyTitle	glob:*Film*Television*

The actual title of the page reached by clicking on the link was "De Anza Film And Television Department - Menu". By using a pattern rather than the exact text, theverifyTitle will pass as long as the two words "Film" and "Television" appear (in that order) anywhere in the page's title.
For example, if the page's owner should shorten the title to just "Film & Television Department," the test would still pass.
Using a pattern for both a link and a simple test that the link worked (such as the verifyTitle above does) can greatly reduce the maintenance for such test cases.

- **Regular Expression Patterns**

Regular expression patterns are the most powerful of the three types of patterns that Selenese supports.
Regular expressions are also supported by most high-level programming languages, many text editors, and a host of tools, including the Linux/Unix command-line utilities **grep**, **sed**, and **awk**.
In Selenese, regular expression patterns allow a user to perform many tasks that would be very difficult otherwise.

For example, suppose your test needed to ensure that a particular table cell contained nothing but a number. regexp: [0-9]+ is a simple pattern that will match a decimal number of any length.
Whereas Selenese globbing patterns support only the ***** and **[ ]** (character class) features, Selenese regular expression patterns offer the same wide array of special characters that exist in JavaScript. Below are a subset of those special characters:

- **Exact Patterns**

The **exact** type of Selenium pattern is of marginal usefulness. It uses no special characters at all. So, if you needed to look for an actual asterisk character (which is special for both globbing and regular expression patterns), the **exact** pattern would be one way to do that. For example, if you wanted to select an item labeled "Real *" from a dropdown, the following code might work or it might not. The asterisk in the glob:Real *pattern will match anything or nothing. So, if there was an earlier select option labeled "Real Numbers," it would be the option selected rather than the "Real *" option.
select//selectglob:Real *In order to ensure that the "Real *" item would be selected, the exact: prefix could be used to create an **exact** pattern as shown below:
select//selectexact:Real *But the same effect could be achieved via escaping the asterisk in a regular expression pattern:
select//selectregexp:Real *It's rather unlikely that most testers will ever need to look for an asterisk or a set of square brackets with characters inside them (the character class for globing patterns). Thus, globing patterns and regular expression patterns are sufficient for the vast majority of us.

Commonly Used Selenium Commands

Name	Description
Open	Open's a page using a URL.
click/clickAndWait	performs a click operation, and optionally waits for a new page to load.
verifyTitle/assertTitle	verifies an expected page title.
verifyTextPresent	verifies expected text is somewhere on the page.
verifyElementPresent	verifies an expected UI element, as defined by its HTML tag, is present on the page.
verifyText	verifies expected text and its corresponding HTML tag are present on the page.
verifyTable	verifies a table's expected contents.
waitForPageToLoad	pauses execution until an expected new page loads. Called automatically when clickAndWait is used.
waitForElementPresent	pauses execution until an expected UI element, as defined by its HTML tag, is present on the page.

Verifying Page Elements

Verifying UI elements on a web page is probably the most common feature of your automated tests. Selenese allows multiple ways of checking for UI elements. It is important that you understand these different methods because these methods define what you are actually testing.

For example, will you test that...

- an element is present somewhere on the page?
- specific text is somewhere on the page?
- specific text is at a specific location on the page?

For example, if you are testing a text heading,the text and its position at the top of the page are probably relevant for your test.

If, however, you are testing for the existence of an image on the home page, and the web designers frequently change the specific image file along with its position on the page, then you only want to test that *an image* (as opposed to the specific image file) exists *somewhere on the page*.

Assertion or Verification?

Choosing between "assert" and "verify" comes down to convenience and management of failures. There's very little point checking that the first paragraph on the page is the correct one if your test has already failed when checking that the browser is displaying the expected page.

If you're not on the correct page, you'll probably want to abort your test case so that you can investigate the cause and fix the issue(s) promptly.

On the other hand, you may want to check many attributes of a page without aborting the test case on the first failure as this will allow you to review all failures on the page and take the appropriate action. Effectively an "assert" will fail the test and abort the current test case, whereas a "verify" will fail the test and continue to run the test case.

The best use of this feature is to logically group your test commands, and start each group with an **"assert" followed by one or more "verify" test commands. An example follows:**

Command	Target	value
open	/download/	
assertTitle	Downloads	
verifyText	//h2	Downloads
assertTable	1.2.1	Selenium IDE
verifyTable	1.2.2	June 3, 2008
verifyTable	1.2.3	1.0 beta 2

The above example first opens a page and then "asserts" that the correct page is loaded by comparing the title with the expected value. Only if this passes will the following command run and "verify" that the text is present in the expected location. The test case then "asserts" the first column in the second row of the first table contains the expected value, and only if this passed will the remaining cells in that row be "verified".

verifyTextPresent

The command verifyTextPresent is used to verify *specific text exists somewhere on the page*. It takes a single argument–the text pattern to be verified. For example:

Command	Target	value
verifyTextPresent	Marketing Analysis	

This would cause Selenium to search for, and verify, that the text string “Marketing Analysis” appears somewhere on the page currently being tested. Use verifyTextPresent when you are interested in only the text itself being present on the page. Do not use this when you also need to test where the text occurs on the page.

- **verifyElementPresent**

Use this command when you must test for the presence of a specific UI element, rather than its content. This verification does not check the text, only the HTML tag. One common use is to check for the presence of an image.

Command	Target	value
verifyElementPresent	//div/p/img	

This command verifies that an image, specified by the existence of an <img> HTML tag, is present on the page, and that it follows a tag and a <p> tag. The first (and only) parameter is a *locator* for telling the Selenese command how to find the element. Locators are explained in the next section.
verifyElementPresent can be used to check the existence of any HTML tag within the page. You can check the existence of links, paragraphs, divisions , etc. Here are a few more examples.

Command	Target	value
verifyElementPresent	//div/p	
verifyElementPresent	//div/a	
verifyElementPresent	id=Login	
verifyElementPresent	link=Go to Marketing Research	
verifyElementPresent	//a[2]	
verifyElementPresent	//head/title	

These examples illustrate the variety of ways a UI element may be tested. Again, locators are explained in the next section.

verifyText

Use verifyText when both the text and its UI element must be tested. verifyText must use a locator. If you choose an *XPath* or *DOM* locator, you can verify that specific text appears at a specific location on the page relative to other UI components on the page.

Command	Target	value
verifyText	//table/tr/td/div/p	This is my text and it occurs right after the div inside the table.

Selenium WebDriver

Selenium suite is comprised of 4 basic components; Selenium IDE, Selenium RC, WebDriver, Selenium Grid.
WebDriver allows user to perform web based automation testing. WebDriver is a different tool altogether that has various advantages over Selenium RC.
WebDriver supports a wide range of web browsers, programming languages and test environments.

WebDriver directly communicates with the web browser and uses its native compatibility to automate. WebDriver's support doesn't only limits in the periphery of traditional user actions. Instead it supports efficient handling mechanisms for complex user actions like dealing with dropdowns, Ajax calls, switching between windows, navigation, handling alerts etc.
WebDriver enables user to perform web based mobile testing. To support the same, WebDriver introduces AndroidDriver and IphoneDriver.
WebDriver is faster than other tools of Selenium Suite because it makes direct calls to browser without any external intervention.

Java Installation

Step 1: Go to Oracle official site – "JAVA download", download Java Platform, Standard Edition. All the recent releases are available on the page.

Step 2: As soon as you click on the Download button, following screen would appear. Accept the License agreement for Java installation and choose amongst the various catalogued Java Development Kit's. Select the one that best suits your system configuration.
Remember to download JDK (Java development kit). The kit comes with a JRE (Java Runtime environment). Thus the user isn't required to download and install the JRE separately.

Eclipse IDE Installation

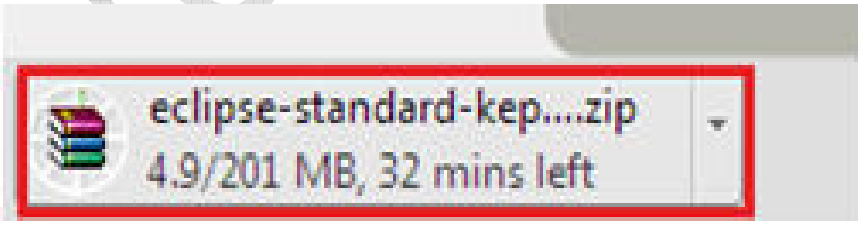
Step 1: Go to Eclipse official website and navigate to its download page – Eclipse download. Download Eclipse IDE for Java EE developers. All the recent releases are available on the page.

Make sure you opt and download the appropriate eclipse IDE as per your system configuration. There are two download links available for 64 bit windows operating system and 32-bit windows operating system.

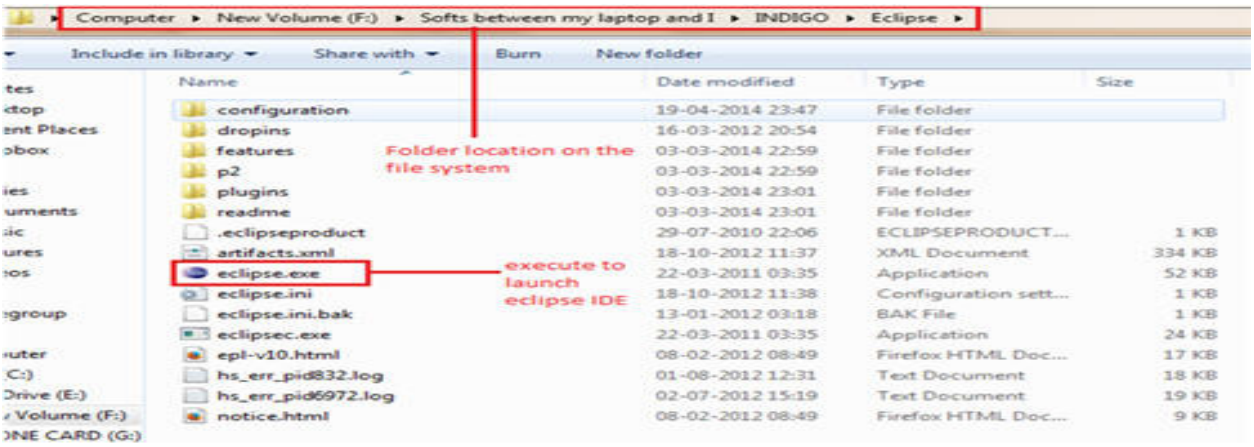
Step 2: As soon as we click on the download link, the user is re-directed to the fresh page securing information about the current download. Click on the download icon and you are done.

It may take a few minutes before you can download the complete zip folder.

Step 3: Once downloaded, copy the folder and place it in the desired location on your file system.

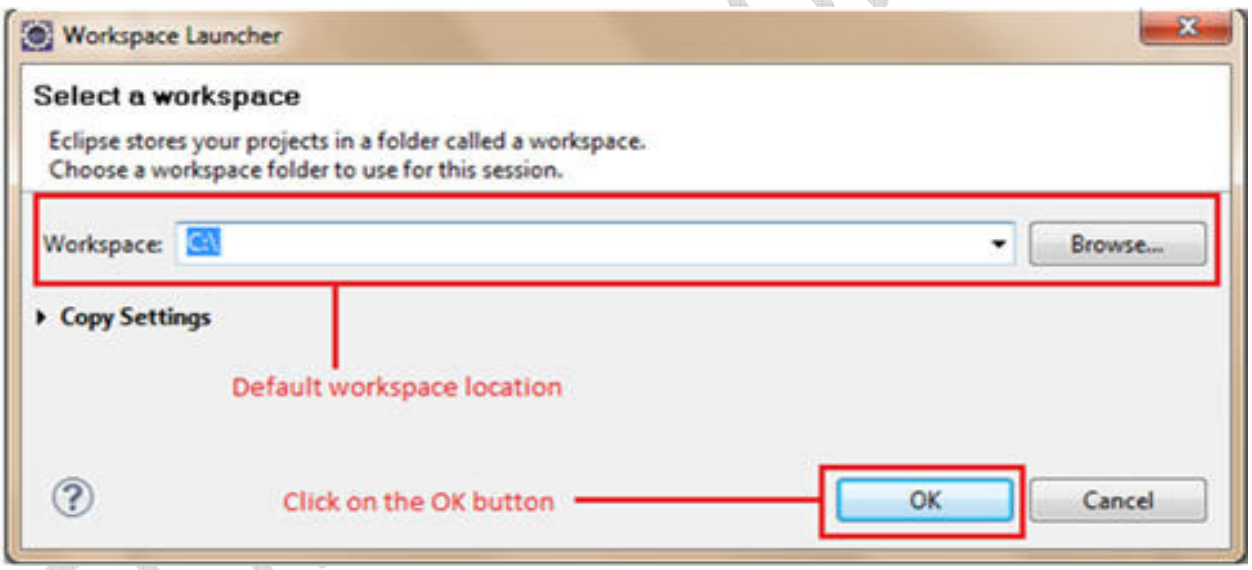


Step 4: Extract the zipped folder, a folder named as eclipse can be seen. The folder embodies all the required application and source files.



Step 5: Launch the eclipse IDE using "eclipse.exe" residing inside the eclipse folder. Refer the above illustration for the same.

Step 6: The application will prompt you to specify the workspace location. Workspace is that location where all your eclipse projects will be residing. Enter/Browse the desired location or the user can simply opt for the default location and click on the OK button.



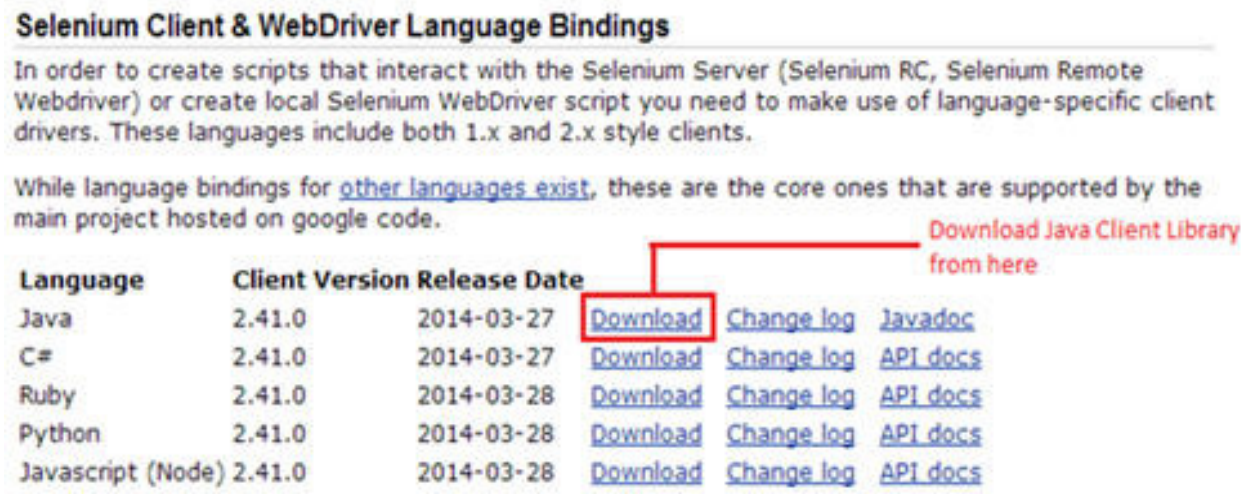
Configuring WebDriver

As we would be using Java as the programming language for this series and in order to create test scripts in java, we would have to introduce language- specific client drivers. Thus, let us begin with the downloading of Selenium Java Client Libraries.

Download the Selenium Java Client Libraries

Step 1: Go to Selenium's official website and navigate to its download page – "http://docs.seleniumhq.org/download/". Refer the section in the below illustration where you can find

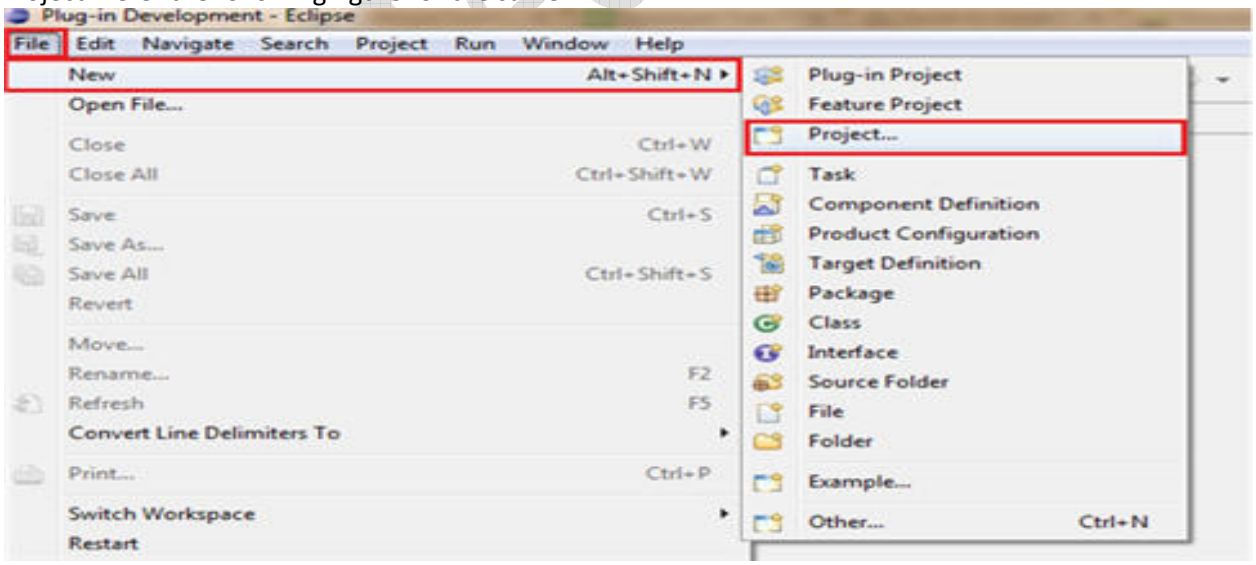
Client Libraries listed for distinct programming languages. Click on the download link for Java Client Library.



It may take a few minutes before you can download the complete zipped folder.
Step 2: Once downloaded, copy the folder and place it in the desired location on your file system.
Step 3: Extract the zipped folder, a folder named as "Selenium-2.41.0.zip"can be seen. The folder embodies all the required jar files which enable users to create test scripts in Java.
Thus these libraries can be configured in Eclipse IDE.

Configuring Libraries with Eclipse IDE

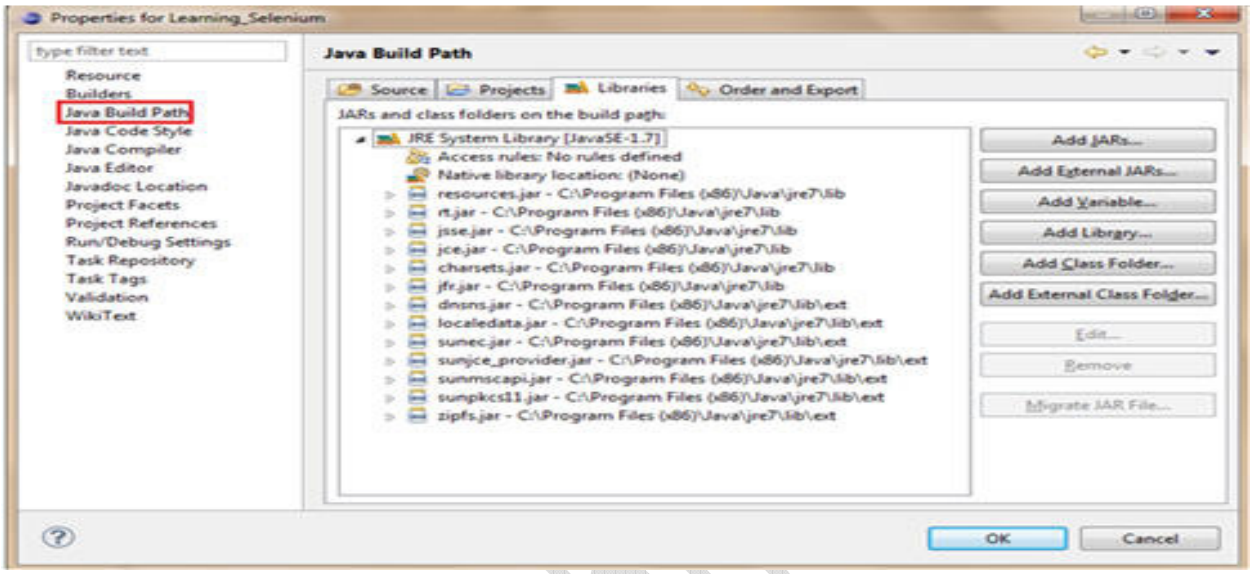
Step 1: Navigate towards Eclipse IDE. Create a new java based project following File -> New -> Java Project. Refer the following figure for the same.



Step 2: Provide a user defined name for your Java Project. Let us provide the name as Learning_Selenium and Click on the Finish Button. The newly created project can be viewed at the left side of the screen in the package explorer panel.

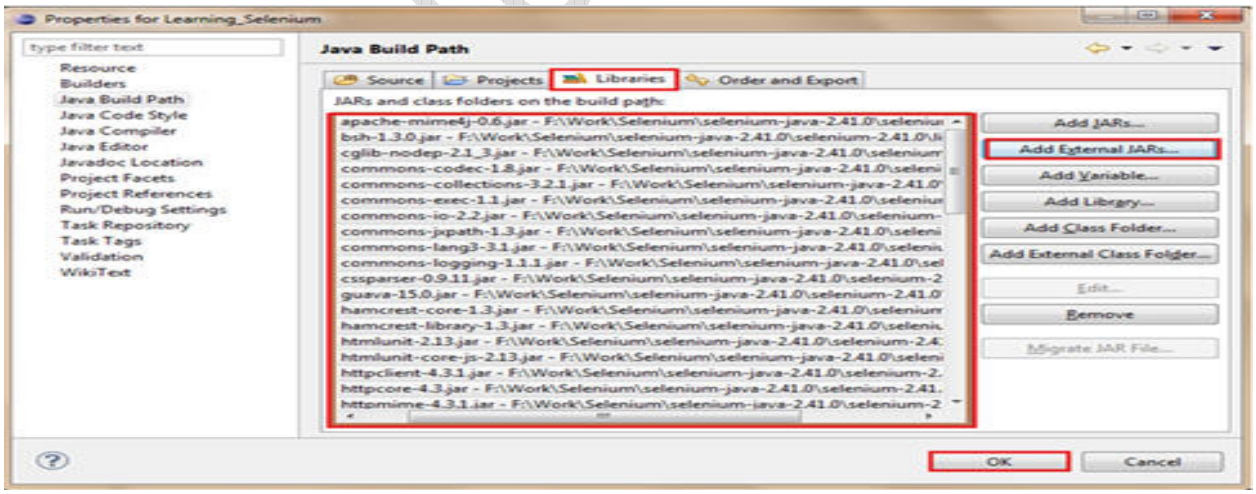
Step 3: Create a new Java class named as “First_WebdriverClass” under the source folder by right clicking on it and navigating to New -> class.

Step 4: Now let us configure the libraries into our Java project. For this, select the project and Right click on it. Select “Properties” within the listed options. The following screen appears, Select “Java Build Path” from the options.



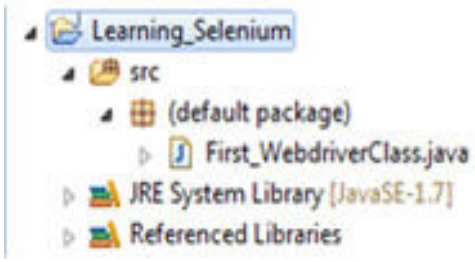
Step 5: By default, “Libraries” tab is opened. If not, click on the “Libraries” tab. Then, click on the “Add External Jars…” button. Browse to the location where we have saved the extracted folder for Java Client Libraries.

Step 6: Select all the JAR files present in the “selenium-java-2.41.0” folder and click on open button within the dialog box. The properties dialog box should look like the below illustration.



Step 7: Click on the “OK” button within the dialog box so as to complete the configuration part of Selenium Libraries in our java project.

The project will look like the following:




```
public class First_WebdriverClass {

}
```

Available Drivers

There are a number of driver classes available in WebDriver, each catering a specific web browser. Each browser has a different driver implementation in WebDriver.

In WebDriver, a few of the browsers can be automated directly where as some of the web browsers require an external entity to be able to automate and execute the test script. This external entity is known as Driver Server. Thus, user is required to download the Driver Server for different web browsers. Notice that there is a separate Driver Server for each of the web browser and user cannot use one Driver Server for web browsers other than the one it is designated for.

Below is the list of available web browsers and their corresponding Server Drivers.

Web-Browser	Driver Server
Mozilla Firefox	No (No external server is required to spin the Firefox browser)
Google Chrome	Yes (ChromeDriver)
Internet Explorer	Yes (Internet Explorer Driver Server)
Opera	Yes (OperaDriver)
Safari	Yes (SafariDriver)
HTML Unit	No (No external entity is required to spin the HTML Unit)

Script Creation

For script creation, we would be using “Learning_Selenium” project created in the previous tutorial and “gmail.com” as the application under test (AUT).

Scenario:

- Launch the browser and open “Gmail.com”.
- Verify the title of the page and print the verification result.
- Enter the username and Password.
- Click on the Sign in button.
- Close the web browser.
- Step 1: Create a new java class named as “Gmail_Login” under the “Learning_Selenium” project.
- Step 2: Copy and paste the below code in the “Gmail_Login.java” class.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
public class Gmail_Login {
    public static void main(String[] args) {
// objects and variables instantiation
        WebDriver driver = new FirefoxDriver();
        String appUrl ="https://accounts.google.com";
// launch the firefox browser and open the application url
        driver.get(appUrl);
// maximize the browser window
        driver.manage().window().maximize();
// declare and initialize the variable to store the expected title of the webpage.
        String expectedTitle = " Sign in - Google Accounts ";
// fetch the title of the web page and save it into a string variable
        String actualTitle = driver.getTitle();
// compare the expected title of the page with the actual title of the page and print the result
        if (expectedTitle.equals(actualTitle)) {
            System.out.println("Verification Successful - The correct title is displayed on the web page.");
} else {
            System.out.println("Verification Failed - An incorrect title is displayed on the web page.");
}
// enter a valid username in the email textbox
        WebElement username = driver.findElement(By.id("Email"));
        username.clear();
        username.sendKeys("TestSelenium");
// enter a valid password in the password textbox
        WebElement password = driver.findElement(By.id("Passwd"));
        password.clear();
        password.sendKeys("password123");
// click on the Sign in button
        WebElement SignInButton = driver.findElement(By.id("signIn"));
        SignInButton.click();
// close the web browser
        driver.close();
        System.out.println("Test script executed successfully.");
// terminate the program
        System.exit(0);
}}
```

Code Walkthrough

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.By;
```

- **Prior to the actual scripting, we need to import the above packages:**

import *org.openqa.selenium.WebDriver* – References the WebDriver interface which is required to instantiate a new web browser.

import *org.openqa.selenium.firefox.FirefoxDriver* – References the FirefoxDriver class that is required instantiate a Firefox specific driver on the browser instance instantiated using WebDriver interface.

import *org.openqa.selenium.WebElement* – References to the WebElement class which is required to instantiate a new web element.

import *org.openqa.selenium.By* – References to the By class on which a locator type is called.

As and when our project would grow, it is evident and logical that we might have to introduce several other packages for more complex and distinct functionalities like excel manipulations, database connectivity, logging, assertions etc.

Object Instantiation

WebDriver driver = new FirefoxDriver();

We create a reference variable for WebDriver interface and instantiate it using FirefoxDriver class. A default Firefox profile will be launched which means that no extensions and plugins would be loaded with the Firefox instance and that it runs in the safe mode.

Launching the Web browser

driver.get(appUrl);

A *get()* method is called on the WebDriver instance to launch a fresh web browser instance. The string character sequence passed as a parameter into the *get()* method redirects the launched web browser instance to the application URL.

Maximize Browser Window

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

The *maximize()* method is used to maximize the browser window soon after it is re-directed to the application URL.

Fetch the page Title

- *driver.getTitle();*

The *getTitle()* method is used to fetch the title of the current web page. Thus, the fetched title can be loaded to a string variable.

- **Comparison between Expected and Actual Values:**

```
if (expectedTitle.equals(actualTitle)) {
System.out.println("Verification Successful - The correct title is displayed on the web page."); }
Else
{
        System.out.println("Verification Failed - An incorrect title is displayed on the web page.");}
```

The above code uses the conditional statement java constructs to compare the actual value and the expected value. Based on the result obtained, the print statement would be executed.

- **WebElement Instantiation**

WebElement username = driver.findElement(By.id("Email"));

In the above statement, we instantiate the WebElement reference with the help of *"driver.findElement(By.id("Email"))".* Thus, username can be used to reference the Email textbox on the user interface every time we want to perform some action on it.

Clear Command

username.clear();

The clear() method/command is used to clear the value present in the textbox if any. It also clears the default placeholder value.

sendKeys Command

username.sendKeys("TestSelenium ");

The *sendKeys()* method/command is used to enter/type the specified value (within the parentheses) in the textbox. Notice that the*sendKeys()* method is called on the WebElement object which was instantiated with the help of element property corresponding to the UI element.

The above block of code enters the string "TestSelenium" inside the Email textbox on the Gmail application.

sendKeys is one of the most popularly used commands across the WebDriver scripts.

Click Command

SignInButton.click();

Like *sendKeys(), click()* is another excessively used command to interact with the web elements. *Click()* command/method is used to click on the web element present on the web page.

The above block of code clicks on the "Sign in" button present on the Gmail application.

Notes:

Unlike sendKeys() method, click() methods can never be parameterized.

At times, clicking on a web element may load a new page altogether. Thus to sustain such cases, click() method is coded in a way to wait until the page is loaded.

- **Close the Web Browser**

driver.close();

The close() is used to close the current browser window.

- **Terminate the Java Program**

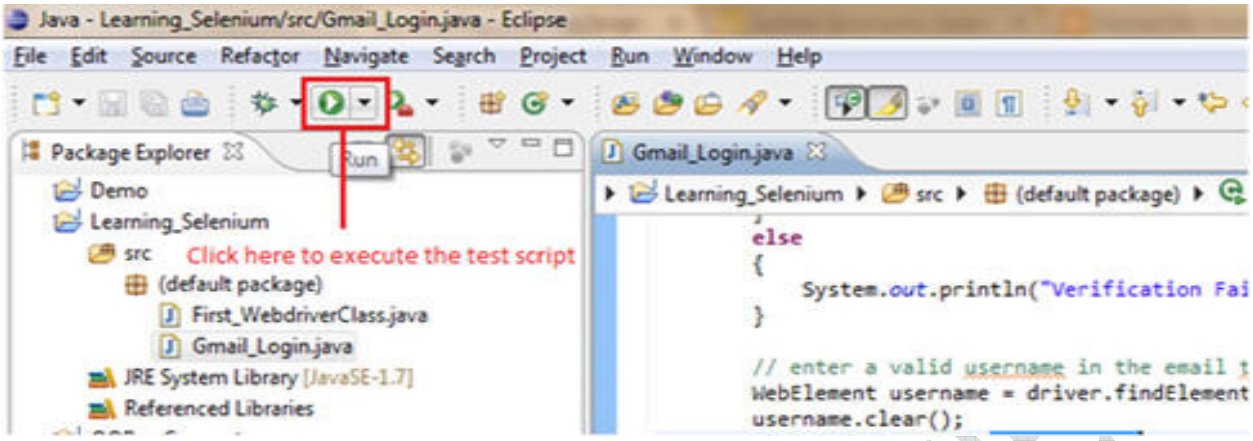
System.exit(0);

The Exit() method terminates the Java program forcefully. Thus, remember to close all the browser instances prior terminating the Java Program.

Test Execution

The test script or simply the java program can be executed in the following ways:

#1. Under the Eclipse's menu bar, there is an icon to execute the test script. Refer the following figure.



Make a note that only the class which is selected would be executed.

#2. Right click anywhere inside the class within the editor, select "Run As" option and click on the "Java Application".

#3. Another shortcut to execute the test script is – Press ctrl + F11.

At the end of the execution cycle, the print statement "Test script executed successfully." can be found in the console.

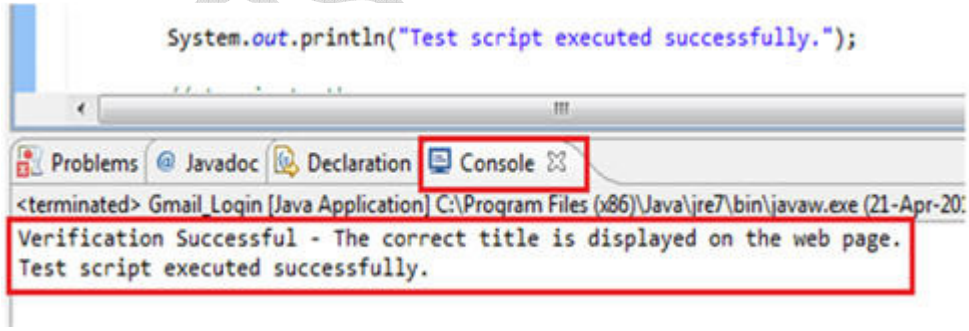
Locating Web Elements

Web elements in WebDriver can be located and inspected in the same way as we did in the previous tutorials of Selenium IDE. Selenium IDE and Firebug can be used to inspect the web element on the GUI. It is highly suggested to use Selenium IDE to find the web elements. Once the web element is successfully found, copy and paste the target value within the WebDriver code. The types of locators and the locating strategies are pretty much the same except for the syntax and their application.

In WebDriver, web elements are located with the help of the dynamic finders (findElement(By.locatorType("locator value"))).

Sample Code:

driver.findElement(By.id("Email"));



Locator Type	Syntax	Dscripition
id	driver.findElement (By.id("ID_of_Element"))	Locate by value of the "id" attribute
Class Name	driver.findElement	Locate by value of

	(By.className ("Class_of_Element"))	the "class" attribute
LinkText	driver.findElement (By.linkText("Text")) .	Locate by value of the text of the hyperlink .
partialLinkText	driver.findElement (By.partialLinkText ("PartialText"))	Locate by value of the sub-text of the hyperlink
name	driver.findElement (By.name ("Name_of_Element"))	Locate by value of the "name" attribute
Xpath	driver.findElement (By.xpath("Xpath"))	Locate by value of the xpath
Cssselector	driver.findElement (By.cssSelector ("CSS Selector"))	Locate by value of the CSS selector
tagname	driver.findElement (By.tagName("input"))	Locate by value of its tag name

- Some Important Commands

Command Name	Description
Get()	It automatically opens a new browser window and fetches the page that you specify inside its parentheses.
getTitle()	Needs no parameters Fetches the title of the current page Leading and trailing white spaces are trimmed Returns a null string if the page has no title .
getPageSource()	Needs no parameters Returns the source code of the page as a String value
getCurrentUrl()	Needs no parameters Fetches the string representing the current URL that the browser is looking at
getText()	Fetches the inner text of the element that you specify .

Navigate commands

CommaND Name	Description
navigate().to()	It automatically opens a new browser window and fetches the page that you specify inside its parentheses. It does exactly the same thing as the get() method

navigate().refresh()	Needs no parameters. It refreshes the current page.
navigate().back()	Needs no parameters Takes you back by one page on the browser's history
navigate().forward()	Needs no parameters Takes you forward by one page on the browser's history
Closing and Quitting Browser Windows .	
close()	Needs no parameters It closes only the browser window that WebDriver is currently controlling.
Quit()	Needs no parameters It closes all windows that WebDriver has opened.

Selenium-WebDriver's Drivers

WebDriver is the name of the key interface against which tests should be written, but there are several implementations. These include:

Driver Name	Description
HtmlUnit Driver	This is currently the fastest and most lightweight implementation of WebDriver. As the name suggests, this is based on HtmlUnit. HtmlUnit is a java based implementation of a WebBrowser without a GUI. For any language binding (other than java) the Selenium Server is required to use this driver WebDriver driver = **new** HtmlUnitDriver();
HtmlUnit Driver	
Pros	Cons
1. Fastest implementation of WebDriver 2. A pure Java solution and so it is platform independent. 3. Supports JavaScript	Emulates other browsers' JavaScript behaviour
Firefox Driver	
	Controls the Firefox browser using a Firefox plugin. The Firefox Profile that is used is stripped down from what is installed on the machine to only include the Selenium WebDriver.xpi (plugin). A few settings are also changed by default (see the source to see which ones) Firefox Driver is capable of being run and is tested on Windows, Mac, Linux. Currently on versions 3.6, 10, latest - 1, latest

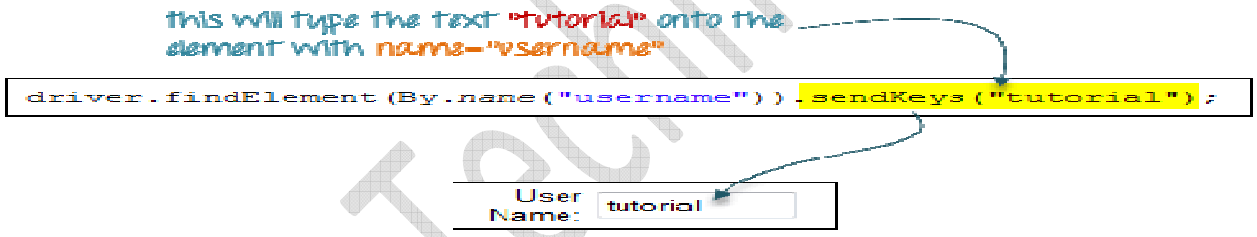
	WebDriver driver = **new** FirefoxDriver();
Pros	Cons
Runs in a real browser and supports JavaScript Faster than the Internet Explorer Driver	Slower than the HtmlUnit Driver
Internet Explorer Driver	
	This driver is controlled by a .dll and is thus only available on Windows OS. Each Selenium release has its core functionality tested against versions 6, 7 and 8 on XP, and 9 on Windows7. WebDriver driver = **new** InternetExplorerDriver();
Pros	Cons
Runs in a real browser and supports JavaScript with all the quirks your end users see.	Obviously the Internet Explorer Driver will only work on Windows! Comparatively slow (though still pretty snappy :) XPath is not natively supported in most versions. Sizzle is injected automatically which is significantly slower than other browsers and slower when comparing to CSS selectors in the same browser. CSS is not natively supported in versions 6 and 7. Sizzle is injected instead. CSS selectors in IE 8 and 9 are native, but those browsers don't fully support CSS3
ChromeDriver	
	ChromeDriver is maintained / supported by the Chromium project iteslf. WebDriver works with Chrome through the chromedriver binary (found on the chromium project's download page). You need to have both chromedriver and a version of chrome browser installed. chromedriver needs to be placed somewhere on your system's path in order for WebDriver to automatically discover it. The Chrome browser itself is discovered by chromedriver in the default installation path. These both can be overridden by environment variables. Please refer to the wiki for more information. WebDriver driver = **new** ChromeDriver();
Pros	Cons
Runs in a real browser and supports JavaScript Because Chrome is a Webkit-based browser, the ChromeDriver may allow you to verify that	Slower than the HtmlUnit Driver

your site works in Safari. Note that since Chrome uses its own V8 JavaScript engine rather than Safari's Nitro engine, JavaScript execution may differ.	

Accessing Forms using Selenium WebDriver

- **Accessing Form Elements**
- **Input Box**
- Input boxes refer to either of these two types:
- **Text Fields**- text boxes that accept typed values and show them as they are.
- **Password Fields**- text boxes that accept typed values but mask them as a series of special characters (commonly dots and asterisks) to avoid sensitive values to be displayed.

Entering Values in Input Boxes The **sendKeys()** method is used to enter values into input boxes.



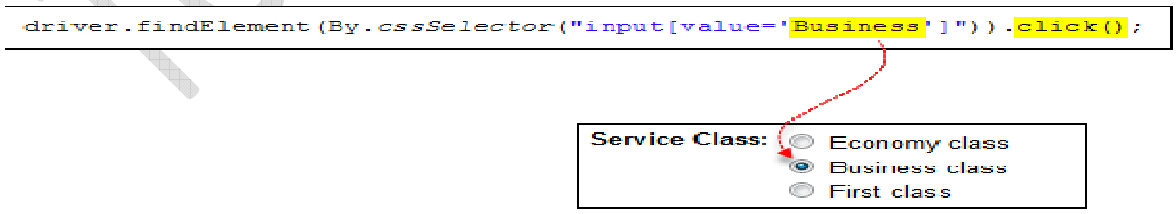
Deleting Values in Input Boxes

The **clear()** method is used to delete the text in an input box. **This method does not need any parameter**. The code snippet below will clear out the text "tutorial" in the User Name text box.

```
driver.findElement(By.name("userName")).clear();
```

Radio Button

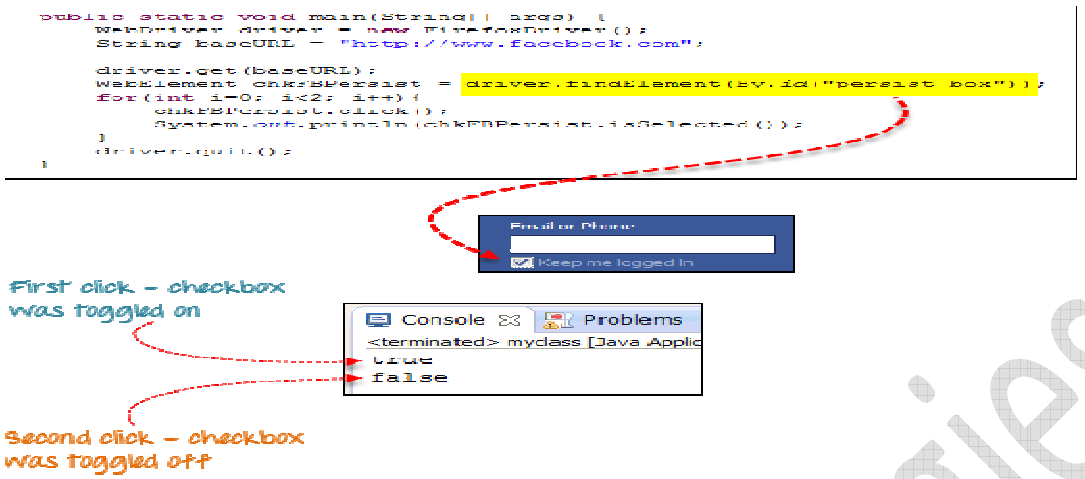
Toggling a radio button on is done using the **click()** method.



- **Check Box**

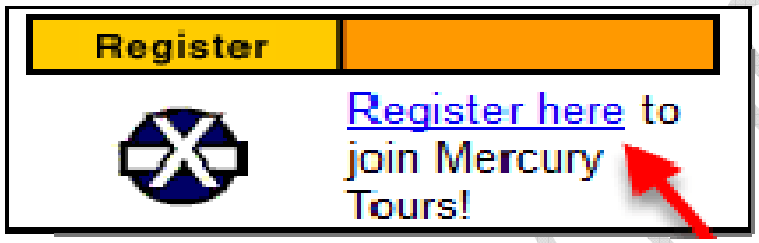
Toggling a check box on/off is also done using the **click()** method.

The code below will click on Facebook's "Keep me logged in" check box twice and then output the result as TRUE when it is toggled on, and FALSE if it is toggled off.



- **Links**

Links also are accessed by using the **click()** method. Consider the below link found in Mercury Tours' homepage.



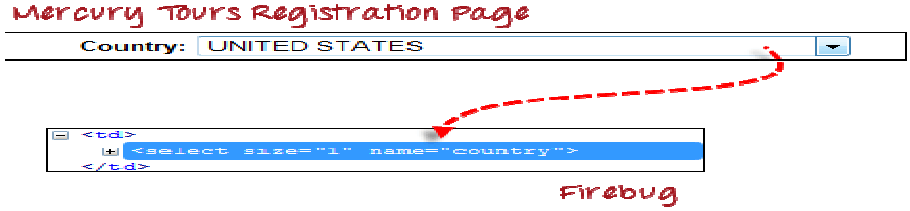
You can access this link using linkText() or partialLinkText() together with click(). Either of the two lines below will be able to access the "Register here" link shown above.

```
driver.findElement(By.linkText("Register here")).click();
```

```
driver.findElement(By.partialLinkText("here")).click();
```

- **Drop-Down Box**

Before we can control drop-down boxes, we must do following two things :
Import the package **org.openqa.selenium.support.ui.Select**
Instantiate the drop-down box as a "Select" object in WebDriver
As an example, go to Mercury Tours' Registration page (http://newtours.demoaut.com/mercuryregister.php) and notice the "Country" drop-down box there.



When submit() is used, WebDriver will look up the DOM to know which form the element belongs to, and then trigger its submit function.

Element	Command	Description
InputBox	*sendKeys()* *clear()*	used to enter values onto text boxes used to clear text boxes of its current value
Check Box, Radio Button,	*Click()*	used to toggle the element on/off
Link	*Clicks*	used to click on the link and wait for page load to complete before proceeding to the next command.
Drop-Down Box	*selectByVisibleText()/* *deselectByVisibleText()* *selectByValue()/* *deselectByValue()* *selectByIndex()/* *deselectByIndex()* *isMultiple()* *deselectAll()*	selects/deselects an option by its displayed text selects/deselects an option by the value of its "value" attribute selects/deselects an option by its index returns TRUE if the drop-down element allows multiple selection at a time; FALSE if otherwise deselects all previously selected options
Submit Button	*Submit()*	

Accessing Links & Tables using Selenium Webdriver

- **Accessing Links**
- **Links Matching a Criterion**

Links can be accessed using an exact or partial match of their link text. The examples below provide scenarios where multiple matches would exist, and would explain how WebDriver would deal with them.

- **Exact Match**

Accessing links using their exact link text is done through the By.linkText() method. However, if there are two links that have the very same link text, this method will only access the first one. Consider the HTML code below

```
<html>
    <head>
        <title>Sample</title>
    </head>
    <body>
        <a href="http://www.google.com">click here</a>
        <br>
        <a href="http://www.fb.com">click here</a>
    </body>
</html>
```

When you try to run the WebDriver code below, you will be accessing the first "click here" link

```
public static void main(String[] args) {
     String baseUrl = "file:///D:/newhtml.html";
     WebDriver driver = new FirefoxDriver();

     driver.get(baseUrl);
     driver.findElement(By.linkText("click here")).click();
     System.out.println("Title of page is: " + driver.getTitle());
     driver.quit();
}
```

- As a result, you will automatically be taken to Google

```
Console
<terminated> myclass (1) [Java Application] C
Title of page is: Google
```

- **Partial Match**

Accessing links using a portion of their link text is done using the **By.partialLinkText()** method. If you specify a partial link text that has multiple matches, only the first match will be accessed. Consider the HTML code below.


```
<html>
     <head>
          <title>Partial Match</title>
     </head>
     <body>
          <a href="http://www.google.com">go here</a>
          <br>
          <a href="http://www.fb.com">click here</a>
     </body>
</html>
```

When you execute the WebDriver code below, you will still be taken to Google.

```
public static void main(String[] args) {
     String baseUrl = "file:///D:/partial_match.html";
     WebDriver driver = new FirefoxDriver();

     driver.get(baseUrl);
     driver.findElement(By.partialLinkText("here")).click();
     System.out.println("Title of page is: " + driver.getTitle());
     driver.quit();
}
```

```
Console
<terminated> myclass (1) [Java Applicatio
Title of page is: Google
```

Case-sensitivity The parameters for **By.linkText()** and **By.partialLinkText()** are both case-sensitive, meaning that capitalization matters. For example, in Mercury Tours' homepage, there are two links that contain the text "egis" - one is the "REGISTER" link found at the top menu, and the other is the "Register here" link found at the lower right portion of the page.

The link at the top menu

The link at the lower right portion of the page

Though both links contain the character sequence "egis", the "By.partialLinkText()" method will access these two links separately depending on capitalization of the characters. See the sample code below.

```
public static void main(String[] args) {
     String baseUrl = "http://newtours.demoaut.com/";
     WebDriver driver = new FirefoxDriver();

     driver.get(baseUrl);

     String theLinkText = driver.findElement(By
               .partialLinkText("egis"))
               .getText();
     System.out.println(theLinkText);
     theLinkText = driver.findElement(By
               .partialLinkText("EGIS"))
               .getText();
     System.out.println(theLinkText);

     driver.quit();
}
```

```
Console
<terminated> myclass (1)
Register here
REGISTER
```

- **All Links**

One of the common procedures in web testing is to test if all the links present within the page are working. This can be conveniently done using a combination of the **Java for-each loop** and the **By.tagName("a")** method. The WebDriver code below checks each link from the Mercury Tours homepage to determine those that are working and those that are still under construction.

```
import java.util.List;
 import java.util.concurrent.TimeUnit;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
public class AllLinks {
   public static void main(String[] args) {
      String baseUrl = "http://newtours.demoaut.com/";
      WebDriver driver = new FirefoxDriver();
      String underConsTitle = "Under Construction: Mercury Tours";
      driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
      driver.get(baseUrl);
      List<WebElement> linkElements = driver.findElements(By.tagName("a"));
      String[] linkTexts = new String[linkElements.size()];
      int i = 0;
//extract the link texts of each link element
      for (WebElement e : linkElements) {
         linkTexts[i] = e.getText();
         i++;
      }

      //test each link
      for (String t : linkTexts) {
         driver.findElement(By.linkText(t)).click();
```

```
        if (driver.getTitle().equals(underConsTitle)) {
            System.out.println("\"" + t + "\""
                    + " is under construction.");
        } else {
            System.out.println("\"" + t + "\""
                    + " is working.");
        }
        driver.navigate().back();
    }
    driver.quit();
}
```

- **Links Outside and Inside a Block**

The latest HTML5 standard allows the <a> tags to be placed inside and outside of block-level tags like , <p>, or <h1>. The "By.linkText()" and "By.partialLinkText()" methods can access a link located outside and inside these block-level elements. Consider the HTML code below.

```
<body>
    <p>
        <a href="http://www.google.com">Inside a block-le
    </p>

    <br>
    <a href="http://www.fb.com">
        <div>
            <span>Outside a block-level tag.</span>
        </div>
    </a>
</body>
```

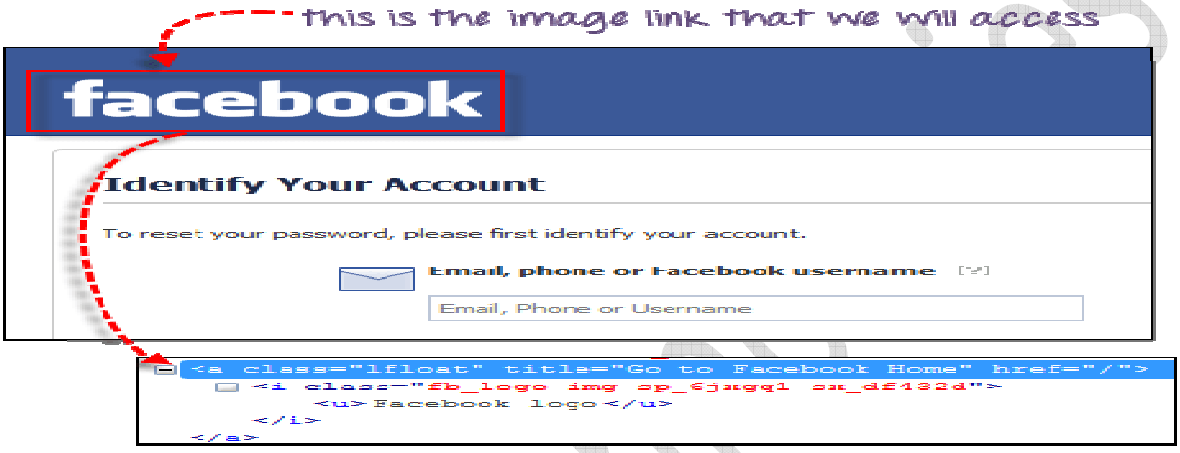
The WebDriver code below accesses both of these links using By.partialLinkText() method.

```
public static void main(String[] args) {
    String baseUrl = "file:///D:/Links%20Outside%20and%20Inside%20a%20Block.html";
    WebDriver driver = new FirefoxDriver();

    driver.get(baseUrl);
    driver.findElement(By.partialLinkText("Inside")).click();
    System.out.println(driver.getTitle());
    driver.navigate().back();
    driver.findElement(By.partialLinkText("Outside")).click();
    System.out.println(driver.getTitle());
    driver.quit();
}
```

- **Accessing Image Links**

Image links are images that act as references to other sites or sections within the same page. Since they are images, we cannot use the By.linkText() and By.partialLinkText() methods because image links basically have no link texts at all. In this case, we should resort to using either By.cssSelector or By.xpath. The first method is more preferred because of its simplicity.

In the example below, we will access the "Facebook" logo on the upper left portion of Facebook's Password Recovery page.



We will use By.cssSelector and the element's "title" attribute to access the image link. And then we will verify if we are taken to Facebook's homepage.

```
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
 public class ImageLink {
    public static void main(String[] args) {
      String baseUrl = "https://www.facebook.com/login/identify?ctx=recover";
      WebDriver driver = new FirefoxDriver();
       driver.get(baseUrl);
      //click on the "Facebook" logo on the upper left portion
      driver.findElement(By.cssSelector("a[title=\"Go to Facebook Home\"]")).click();
 //verify that we are now back on Facebook's homepage
      if (driver.getTitle().equals("Welcome to Facebook - Log In, Sign Up or Learn More")) {
         System.out.println("We are back at Facebook's homepage");
      } else {
         System.out.println("We are NOT in Facebook's homepage");
      }
      driver.quit();
   }
}
```

- **Reading a Table**

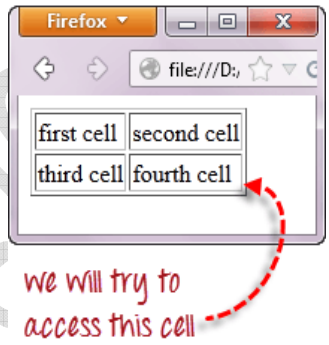
There are times when we need to access elements (usually texts) that are within HTML tables. However, it is very seldom for a web designer to provide an id or name attribute to a certain cell in the table.

Therefore, we cannot use the usual methods such as "By.id()", "By.name()", or "By.cssSelector()". In this case, the most reliable option is to access them using the "By.xpath()" method.

- **XPath Syntax**

Consider the HTML code below.

```
<html>
    <head>
        <title>Sample</title>
    </head>
    <body>
        <table border="1">
            <tbody>
                <tr>
                    <td>first cell</td>
                    <td>second cell</td>
                </tr>
                <tr>
                    <td>third cell</td>
                    <td>fourth cell</td>
                </tr>
            </tbody>
        </table>
    </body>
</html>
```



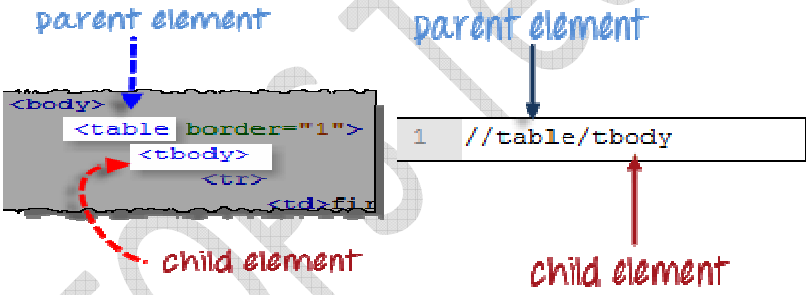
We will use XPath to get the inner text of the cell containing the text "fourth cell".

Step 1 - Set the Parent Element (table)

XPath locators in WebDriver always start with a double forward slash "//" and then followed by the parent element. Since we are dealing with tables, the parent element should always be the <table> tag. The first portion of our XPath locator should therefore start with "//table".

Step 2 - Add the child elements

The element immediately under <table> is <tbody> so we can say that <tbody> is the "child" of <table>. And also, <table> is the "parent" of <tbody>. All child elements in XPath are placed to the right of their parent element, separated with one forward slash "/" like the code shown below.



- **Step 3 - Add Predicates**

The <tbody> element contains two <tr> tags. We can now say that these two <tr> tags are "children" of <tbody>. Consequently, we can say that <tbody> is the parent of both the <tr> elements.

Another thing we can conclude is that the two <tr> elements are siblings. **Siblings refer to child elements having the same parent**.

To get to the <td> we wish to access (the one with the text "fourth cell"), we must first access the **second** <tr> and not the first. If we simply write "//table/tbody/tr", then we will be accessing the first <tr> tag.

So, how do we access the second <tr> then? The answer to this is to use **Predicates**.

Predicates are numbers or HTML attributes enclosed in a pair of square brackets "[ ]" that distinguish a child element from its siblings. Since the <tr> we need to access is the second one, we shall use "[2]" as the predicate.

```
1  //table/tbody/tr[2]
```

The [2] predicate denotes that we are accessing the 2nd <tr> of the parent <tbody>

If we won't use any predicate, XPath will access the first sibling. Therefore, we can access the first <tr> using either of these XPath codes.

```
//table/tbody/tr
```

this will automatically access the first <tr> because no predicate was used

```
//table/tbody/tr[1]
```

this will access the first <tr> because the predicate [1] explicitly says it

Step 4 - Add the Succeeding Child Elements Using the Appropriate Predicates

The next element we need to access is the second <td>. Applying the principles we have learned from steps 2 and 3, we will finalize our XPath code to be like the one shown below.

```
//table/tbody/tr[2]/td[2]
```

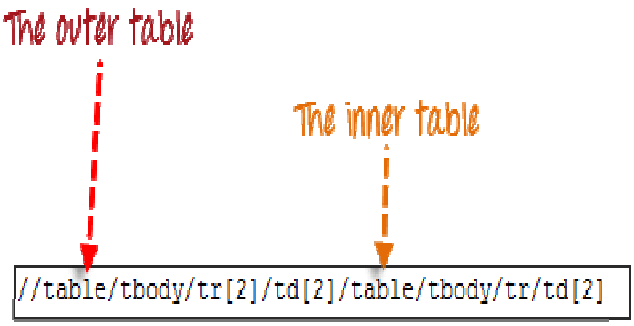
Now that we have the correct XPath locator, we can already access the cell that we wanted to and obtain its inner text using the code below. It assumes that you have saved the HTML code above as "newhtml.html" within your C Drive.

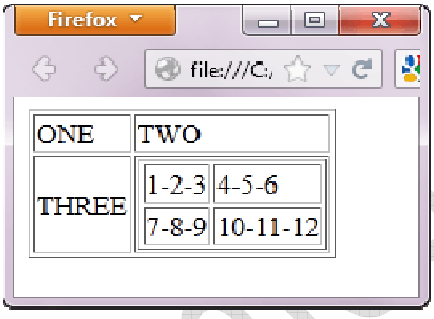
```
    public static void main(String[] args) {
        String baseUrl = "file:///C:/newhtml.html";
        WebDriver driver = new FirefoxDriver();

        driver.get(baseUrl);
        String innerText = driver.findElement(
                By.xpath("//table/tbody/tr[2]/td[2]")).getText();
        System.out.println(innerText);
        driver.quit();
    }
}
```

Accessing Nested Tables

The same principles discussed above applies to nested tables. **Nested tables are tables located within another table**. An example is shown below.

To access the cell with the text "4-5-6" using the "//parent/child" and predicate concepts from the previous section, we should be able to come up with the XPath code below.

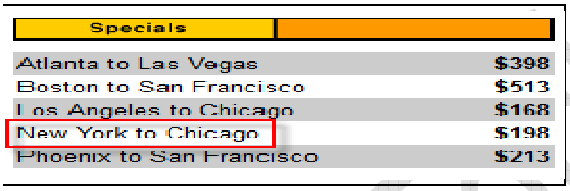




Using Attributes as Predicates

If the element is written deep within the HTML code such that the number to use for the predicate is very difficult to determine, we can use that element's unique attribute instead.

In the example below, the "New York to Chicago" cell is located deep into Mercury Tours homepage's HTML code.



Specials	
Atlanta to Las Vegas	$398
Boston to San Francisco	$513
Los Angeles to Chicago	$168
New York to Chicago	$198
Phoenix to San Francisco	$213

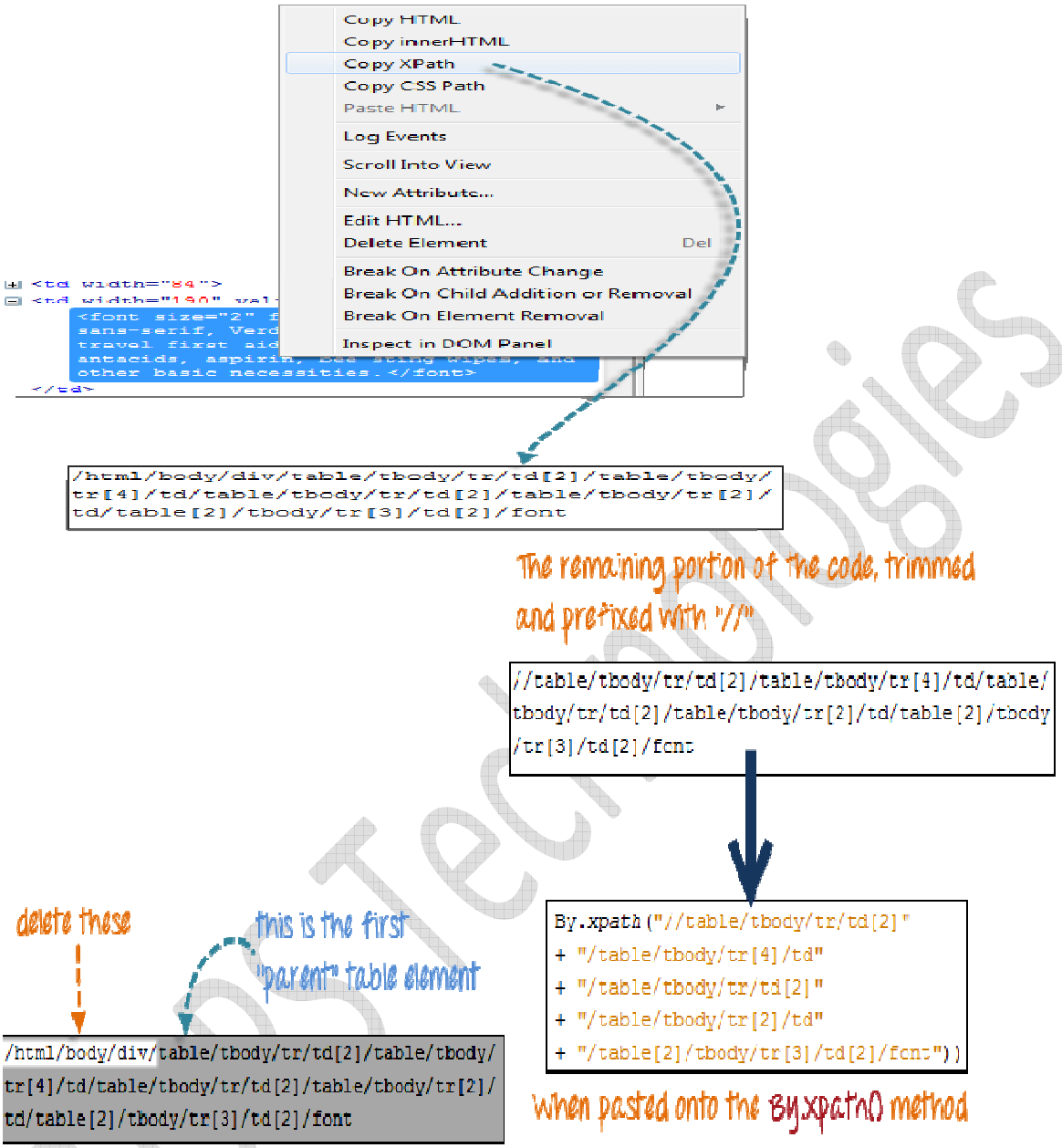
In this case, we can use the table's unique attribute (width="270") as the predicate. **Attributes are used as predicates by prefixing them with the @ symbol**. In the example above, the "New York to Chicago" cell is located in the first <td> of the fourth <tr>, and so our XPath should be as shown below.

```
By.xpath("//table[@width=\"270\"]/tbody/tr[4]/td"))
```

use the escape characters here

Shortcut: Use Firebug

If the number or attribute of an element is extremely difficult or impossible to obtain, the quickest way to generate the XPath code is thru Firebug.



How to write Dynamic XPath in Selenium

Method	Description
Relative XPath method	

Using single attribute	// tagname[@attribute-name='value1']
	Example
	// a [@href='http://www.google.com']
	//input[@id='name']
	//input[@name='username']
	//img[@alt='sometext']

Using multiple attribute	//tagname[@attribute1='value1'][attribute2='value2']

	//a[@id='id1'][@name='namevalue1'] //img[@src=''][@href='']

Using contains method	Syntax
	//tagname[contains(@attribute,'value1')]
	//input[contains(@id,'')]
	//input[contains(@name,'')]
	//a[contains(@href,'')]
	//img[contains(@src,'')]
	//div[contains(@id,'')]

Using starts-with method	//tagname[starts-with(@attribute-name,'')]
	//id[starts-with(@id,'')]
	//a[starts-with(@href='')]
	//img[starts-with(@src='')]
	//div[starts-with(@id='')]
	//input[starts-with(@id='')]
	//button[starts-with(@id,'')]
	And so on.

Using Following node	Xpath/following::again-ur-regular-path
	//input[@id='']/following::input[1]
	//a[@href='']/following::a[1]
	//img[@src='']/following::img[1]

Using preceding node	Xpath/preceding::again-ur-regular-path
	//input[@id='']/ preceding::input[1]
	//a[@href='']/ preceding::a[1]
	//img[@src='']/ preceding::img[1]

Selenium Time out and Synchronization, Wait

Synchronization or wait helps to handle dependencies while executing the script because sometime tools execution speed does not match with the application speed or some web elements response time does not match with script actions.

So to handle synchronization, selenium webdriver provides two effective ways and those ares-

- Implicitly Wait
- Explicitly Wait

Implicitly Way	An implicitly Wait tell to WebDriver to hold for a certain amount of time when trying to find an element or elements if they are not immediately available. The default setting is 0. Once set, the implicit wait is set for the life of the WebDriver object instance. It is a mechanism which will be written once and applied for entire session automatically. It should be applied immediately once we initiate the Webdriver. Implicit wait will not work all the commands/statements in the application.

	It will work only for "FindElement" and "FindElements" statements. If we set implicit wait, find element will not throw an exception if the element is not found in first instance, instead it will poll for the element until the timeout and then proceeds further. We should always remember to add the below syntax immediately below the Webdriver statement. Syntax: driver.manage.TimeOuts.implicitwait(6,Timeunit.SECONDS); **Example:** **WebDriver driver = new FirefoxDriver();** **driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);** **driver.get("www.google.com");**
Explicit Wait	Explicit Wait is intelligent wait and for a particular Web Element. Using explicit wait we basically instruct the wait to x unit of time span before giving up and along with it we can include condition that if particular webelement appear with in x unit of time span then move to next line of code Explicit wait is mostly used when we need to Wait for a specific content/attribute change after performing any action, like when application gives AJAX call to system and get dynamic data and render on UI Example: Like there are drop-downs Country and State, based on the country value selected, the values in the state drop-down will change, which will take few seconds of time to get the data based on user selection. Example WebDriverWait **wait** = new WebDriverWait(driver, 10); **wait**.**until**(ExpectedConditions.visibilityOfElementLocated(By.id("statedropdown"))); OR WebDriverWait load = new WebDriverWait(driver, TimeSpan.FromSeconds(30)); load.Until(ExpectedConditions.ElementExists(By.Id("sb_form_go")))

Dynamic Functions

Setup logging

JDBC Testing using Selenium

Reporting

Working with Calender

If we look at the Datepicker, it is just a like a table with set of rows and columns. To select a date ,we just have to navigate to the cell where our desired date is present.

```
System.setProperty("webdriver.firefox.bin", "C:\\Program Files (x86)\\Mozilla Firefox\\firefox.exe");
 driver = new FirefoxDriver();
  driver.get("http://jqueryui.com/datepicker/");
  driver.switchTo().frame(0);
  driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);
  //Click on textbox so that datepicker will come
  driver.findElement(By.id("datepicker")).click();
```

```
 driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS);
driver.findElement(By.xpath(".//*[@id='ui-datepicker-div']/div/a[2]/span")).click();
    /*DatePicker is a table.So navigate to each cell
  * If a particular cell matches value 13 then select it    */
 WebElement dateWidget = driver.findElement(By.id("ui-datepicker-div"));
 List<webelement> rows=dateWidget.findElements(By.tagName("tr"));
 List<webelement> columns=dateWidget.findElements(By.tagName("td"));
   for (WebElement cell: columns){
  //Select 13th Date
  if (cell.getText().equals("13")){
  cell.findElement(By.linkText("13")).click();
  break;
  }  }
```

Drag and Drop component in Selenium Web Driver

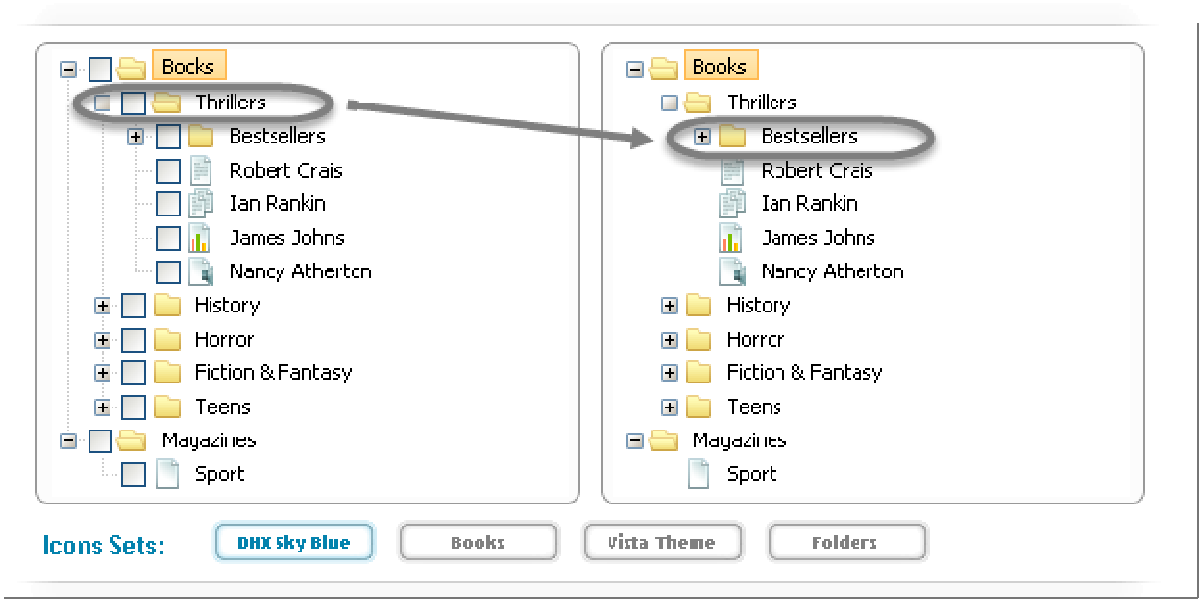
we have a web application where we need to drag an item from one location to another location. These kinds of complex actions are not available in basic element properties.

Automating rich web application is interesting, as it involves advanced user interactions. Selenium has provided a separate “Actions” class to handle these advanced user interactions.

How it works: The action chain generator implements the **Builder** pattern to create a Composite Action containing a group of other actions. This should ease building actions by configuring an **Actions** chains generator instance and invoking its **build( )** method to get the complex action.

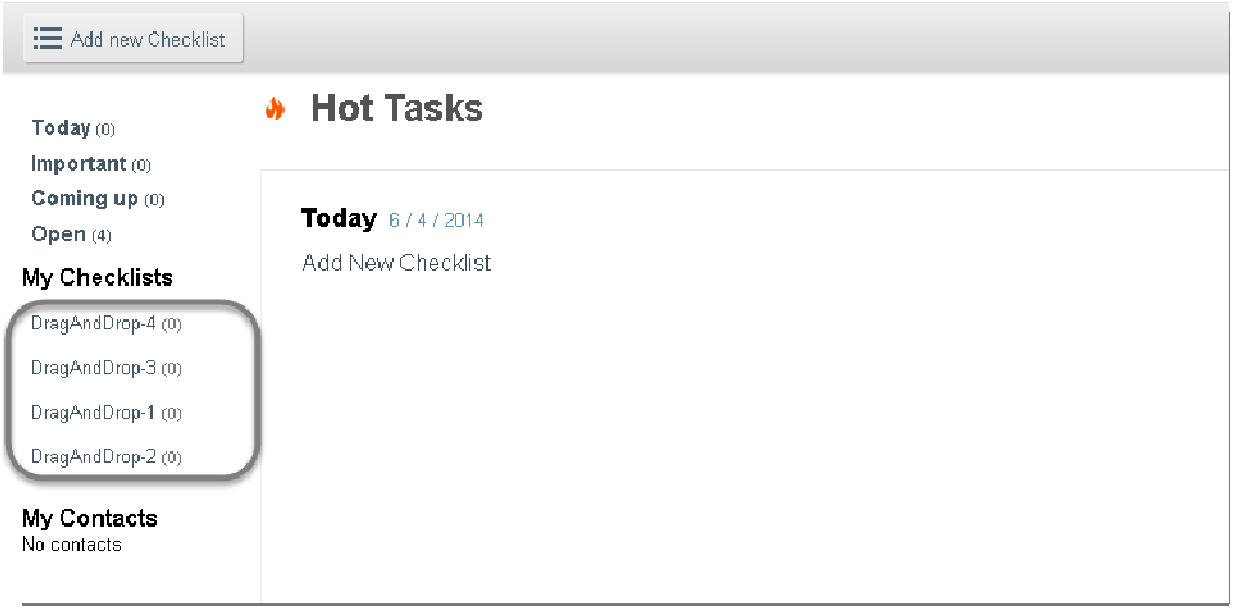
It was really very hard for me to search for a website where I can try ‘Drag n drop’ feature of WebDriver. I was lucky to find two website.

Example 1: In this example we will drag the **Thriller** folder from the left table on to the **Bestsellers** folder of the right side table.



```
WebDriver driver = new FirefoxDriver();
String URL = "http://www.dhtmlx.com/docs/products/dhtmlxTree/index.shtml";
driver.get(URL);
// It is always advisable to Maximize the window before performing DragNDrop action
driver.manage().window().maximize();
driver.manage().timeouts().implicitlyWait(10000, TimeUnit.MILLISECONDS);
WebElement From =
driver.findElement(By.xpath(".//*[@id='treebox1']/div/table/tbody/tr[2]/td[2]/table/tbody/tr[2]/td[2]/table/tbody/tr[1]/td[4]/span"));
WebElement To =
driver.findElement(By.xpath(".//*[@id='treebox2']/div/table/tbody/tr[2]/td[2]/table/tbody/tr[2]/td[2]/table/tbody/tr[2]/td[2]/table/tbody/tr[1]/td[4]/span"));
Actions builder = new Actions(driver);
Action dragAndDrop = builder.clickAndHold(From)
.moveToElement(To)
.release(To)
.build();
dragAndDrop.perform();
```

Example 2: In this example there is a **Checklist** on the left side and there are four sub menus with the name of DragAndDrop-[1-3]. We will login to this website and then drag DragAndDrop-1 to DragAndDrop-4 of the left side sub menu.




```
WebDriver driver = new FirefoxDriver();
                                        String URL = "http://sandbox.checklist.com/account/";
                                        driver.get(URL);
driver.findElement(By.name("j_username")).sendKeys("Username");
```

```
 driver.findElement(By.name("j_password")).sendKeys("Password);
                                                driver.findElement(By.name("login")).click();
                                                 driver.manage().window().maximize();
                                                 driver.manage().timeouts().implicitlyWait(10000,
TimeUnit.MILLISECONDS);
WebElement From = driver.findElement(By.xpath(".//*[@id='userChecklists']/li[1]/a/span"));
                                                 WebElement To =
driver.findElement(By.xpath(".//*[@id='userChecklists']/li[4]/a/span"));
                                                 Actions builder = new Actions(driver);
                                               Action dragAndDrop = builder.clickAndHold(From)
.moveToElement(To)
                                                  .release(To)
                                                 .build();
                                                 dragAndDrop.perform();
```

Keyboard Mouse Events , Uploading Files - Webdriver

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.

Action	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.
sendKeys(onElement, charsequence)	Sends a series of keystrokes onto the element. **Parameters**: *onElement* - element that will receive the keystrokes, usually a text field *charsequence* - any string value representing the sequence of keystrokes to be sent.


```
String baseUrl = "http://newtours.demoaut.com/";
    WebDriver driver = new FirefoxDriver();

    driver.get(baseUrl);
    WebElement link_Home = driver.findElement(By.linkText("Home"));
    WebElement td_Home = driver
        .findElement(By
        .xpath("//html/body/div"
        + "/table/tbody/tr/td"
        + "/table/tbody/tr/td"
```

```
        + "/table/tbody/tr/td"
        + "/table/tbody/tr"));Actions builder = new Actions(driver);

    Action mouseOverHome = builder
        .moveToElement(link_Home)
        .build();


    String bgColor = td_Home.getCssValue("background-color");
    System.out.println("Before hover: " + bgColor);
    mouseOverHome.perform();
    bgColor = td_Home.getCssValue("background-color");
    System.out.println("After hover: " + bgColor);
    driver.quit();
```

Uploading Files

Uploading files in WebDriver is done by simply using the sendKeys() method on the file-select input field to enter the path to the file to be uploaded.

```
String baseUrl = "http://www.megafileupload.com/";
WebDriver driver = new FirefoxDriver();
 driver.get(baseUrl);
WebElement uploadElement = driver.findElement(By.id("uploadfile_0"));
 // enter the file path onto the file-selection input field
uploadElement.sendKeys("C:\\newhtml.html");
 // check the "I accept the terms of service" check box
driver.findElement(By.id("terms")).click();
 // click the "UploadFile" button
driver.findElement(By.name("send")).click();
```

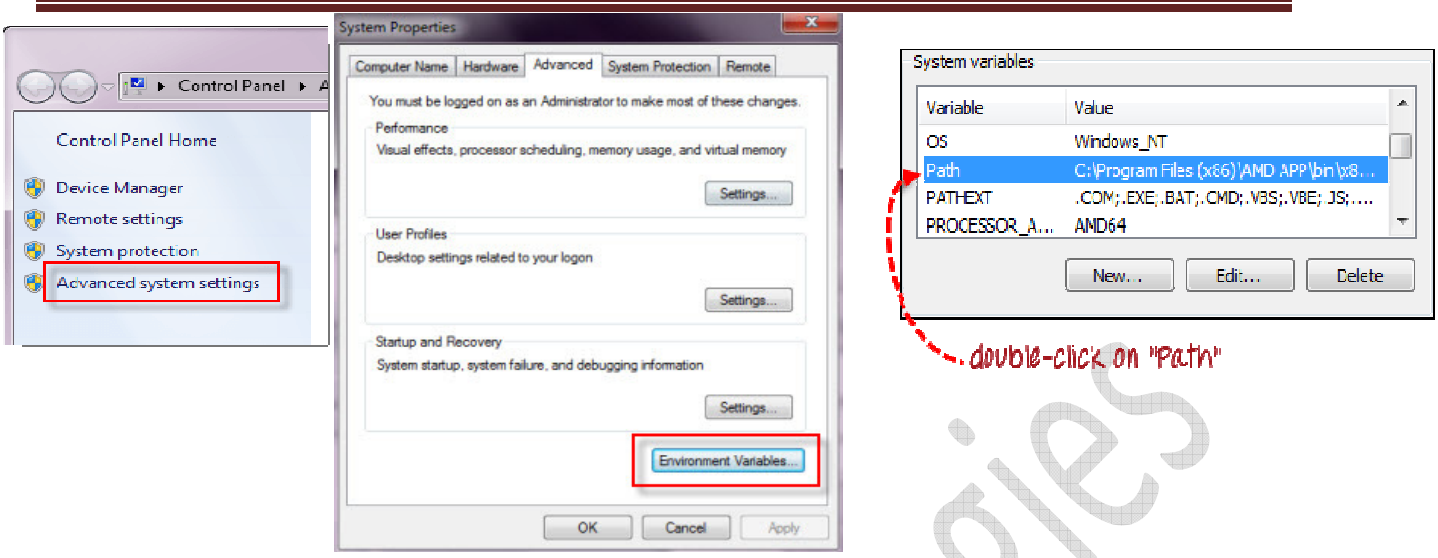
Downloading Files

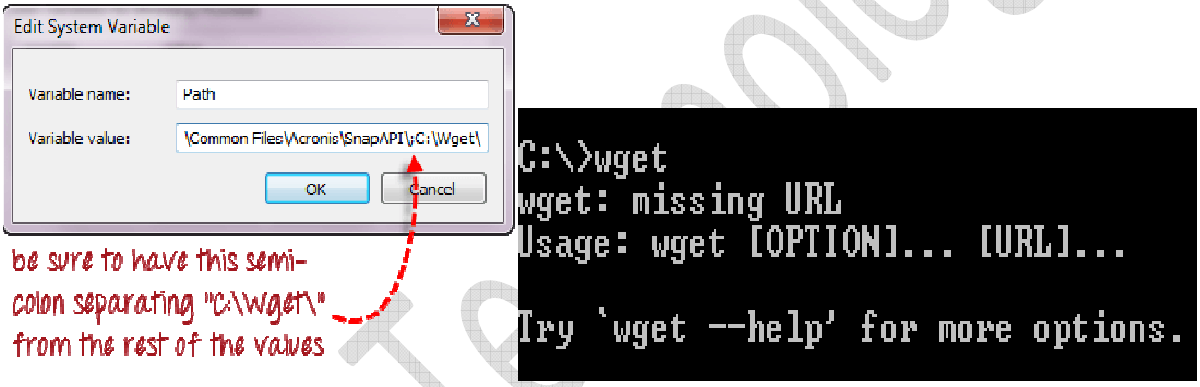
WebDriver has no capability to access the Download dialog boxespresented by browsers when you click on a download link or button. However, we can bypass these dialog boxes using a separate program called "wget".

What is Wget?

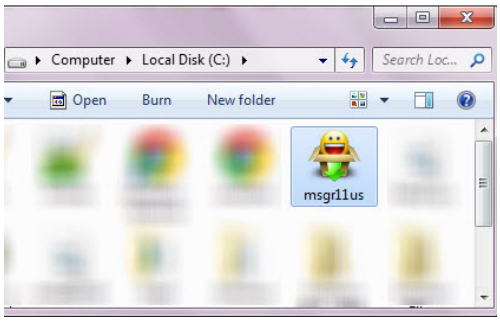
Wget is a small and easy-to-use command-line program used to automate downloads. Basically, we will access Wget from our WebDriver script to perform the download process.

- **Setting up Wget**






```
String baseUrl = "http://messenger.yahoo.com/";
WebDriver driver = new FirefoxDriver();
driver.get(baseUrl);
WebElement downloadButton = driver.findElement(By
.id("messenger-download"));
String sourceLocation = downloadButton.getAttribute("href");
String wget_command = "cmd /c wget -P c: --no-check-certificate " +
sourceLocation;
try {
Process exec = Runtime.getRuntime().exec(wget_command);
int exitVal = exec.waitFor();
System.out.println("Exit value: " + exitVal);
} catch (InterruptedException | IOException ex) {
System.out.println(ex.toString());
}
driver.quit();
}
```



Handle JavaScript, Dialogs

If you want to run JavaScript code in Java WebDriver, do this

WebDriver driver = new AnyDriverYouWant(); if (driver instanceof JavascriptExecutor) {
((JavascriptExecutor)driver).executeScript("yourScript();"); }

Or

WebDriver driver = new AnyDriverYouWant(); JavascriptExecutor js; if (driver instanceof JavascriptExecutor) { js = (JavascriptExecutor)driver; } js.executeScript("return document.getElementById('someId');");

Popup Dialogs

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

Example

```
WebDriver driver = new FirefoxDriver();
    String alertMessage = "";
    driver.get("http://jsbin.com/usidix/1");
    driver.findElement(By.cssSelector("input[value=\"Go!\"]")).click();
    alertMessage = driver.switchTo().alert().getText();
    driver.switchTo().alert().accept();
    System.out.println(alertMessage);
    driver.quit();
```

Log4J Logging

During the running of test case user wants some information to be logged in the console.
Information could be any detail depends upon the purpose.
Keeping this in mind that we are using Selenium for testing, we need the information which helps the User to understand the test steps or any failure during the test case execution.
With the help of Log4j it is possible to enable loggings during the Selenium test case execution for e.g. let's say you have encountered a failure in automation test script and it has to be reported in the system. The set of information that you have required to report a bug is :
A complete test steps to replicate the scenario
Issue, Description of the failure or reason for the failed test case
Time stamp for the developers to investigate the issue in detail

Logging inside the Methods

Logging inside the testcase is very tedious task and sooner or later you will find it boring and annoying. Plus everybody has their own way of writing log messages and messages can be less informative and confusing. So why not make it universal. Writing logs message inside the methods is much helpful way, with that we can avoid lots of confusions, save lot of time and maintain consistency.

- Steps

1) Download JAR files of Log4j and Add Jars to your project library. You can download it from here. That's all about configuration of Apache POI with eclipse. Now you are ready to write your test.

2) Create a new **XML** file – **log4j.xml** and place it under the Project root folder.

3) Paste the following code in the **log4j.xml** file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/" debug="false">
<appender name="fileAppender" class="org.apache.log4j.FileAppender">
<param name="Threshold" value="INFO" />
<param name="File" value="logfile.log"/>
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="%d %-5p [%c{1}] %m %n" />
</layout>
</appender>
<root>
<level value="INFO"/>
<appender-ref ref="fileAppender"/>
</root></log4j:configuration>
```

4) To achieve that we need to create a static **Log** class, so that we can access its log method in any of our project class. Log class will look like this:

```
package utility;
import org.apache.log4j.Logger;
 public class Log {
// Initialize Log4j logs
                                                                    private static
Logger Log = Logger.getLogger(Log.class.getName());//
 // This is to print log for the beginning of the test case, as we usually run so many test cases as a test
suite
 public static void startTestCase(String sTestCaseName){
Log.info("**********************************************************************
**********");
Log.info("**********************************************************************
**********");
Log.info("$$$$$$$$$$$$$$$$$$$$              "+sTestCaseName+ "      $$$$$$$$$$$$$$$$$$$$$$$$$");
Log.info("**********************************************************************
**********");
Log.info("**********************************************************************
**********");
}
                                                                    //This is to
print log for the ending of the test case
 public static void endTestCase(String sTestCaseName){
```

```
                                                                    Log.info("XXX
XXXXXXXXXXXXXXXXXXXX        "+"-E---N---D-"+        XXXXXXXXXXXXXXXXXXXXX");
                                                                    Log.info("X");
                                                                    Log.info("X");
                                                                    Log.info("X");
                                                                    Log.info("X");
                                                                    }
```

5) Insert log messages in **Home_Page** class

Log.info("My Account link element found");

or

Log.info("Log Out link element found");

JDBC with Web Driver

Web Driver cannot directly connect to Database.

You can only interact with your Browser using Web Driver.

For this we use JDBC("Java Database Connectivity").The JDBC API is a Java API for accessing virtually any kind of tabular data.The value of the JDBC API is that an application can access virtually any data source and run on any platform with a Java Virtual Machine.

In simplest terms, a JDBC technology-based driver ("JDBC driver") makes it possible to do three things:

1.Establish a connection with a data source

2.Send queries and update statements to the data source

3.Process the results

1.Establish a connection with a data source

The traditional way to establish a connection with a database is to call the method

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

URL : jdbc:<subprotocol>:<subname>

For connecting to MYSQL URL will be

jdbc:mysql://localhost:3306/hoale

2.Send queries and update statements to the data source

A Statement object is used to send SQL statements to a database over the created connection in Step 1. Statement-created by the *Connection.createStatement* methods. A Statement object is used for sending SQL statements with no parameters.

3.Process the results

A ResultSet is a Java object that contains the results of executing an SQL query.We will have separate post on it.The JDBC API provides three interfaces for sending SQL statements to the database

```
driver = new FirefoxDriver();
    url ="file:///D:/ECLIPSE/workspace_eclipseclassic/ConnectDB/src/com/modules/HTML5Demo.html";
    driver.get(url);
 //Prepare connection
        String url1 ="jdbc:mysql://localhost:3306/hoale";
        // Load Microsoft SQL Server JDBC driver
```

```
        String dbClass = "com.mysql.jdbc.Driver";
        Class.forName(dbClass).newInstance();
        //Get connection to DB
        Connection con = DriverManager.getConnection(url1, "root", "");
        //Create Statement
        Statement stmt = (Statement) con.createStatement();
// method which returns the requested information as rows of data
        ResultSet result = (ResultSet) stmt.executeQuery("select * from employee");
        if(result.next())
        {
            String id = result.getString("ID");
            String info = result.getString("Info");
            driver.getCurrentUrl();
            WebElement a = driver.findElement(By.id("txtID"));
            a.sendKeys(id);
            WebElement b = driver.findElement(By.id("txtInfo"));
            b.sendKeys(info);
            WebElement btnclick = driver.findElement(By.id("btnclick"));
            btnclick.click();
            System.out.print("Passed");
        }
```

Reporting

Reports / Executed Results

Test report/results document which contains summary of test activities performed with pass/fail status and the time taken for execution.

After completing the execution, it is very important to communicate the test results and findings to the project manager and with that decisions can be made for the release.

We can also capture and store all the screen shots for failed scenarios and it also store the executed data sheets with Pass/Fail status.

- Junit
- What is Unit testing?

Unit testing is a *testing* methodology where *individual units are tested* in isolation from other units. This is usually done by ***developers***. A unit can be considered as a class or methods inside a class which needs to be tested individually. It is also know as ***White Box*** Testing, as developer is able to see the code for the functionality.

- What is Junit?
- To do *Unit testing* in *Java* we have an excellent framework called ***Junit***. Junit is a *unit testing framework*. This frameworks provides us with following facilities
  – Base classes and Annotations to write unit tests
  – Base class support to run tests, TestRunner class.

– Base class and Annotation support to write test suites, @RunWith(Suite.Class)
– And of course reporting of test results.

- This tutorial has following sections
- Downloading Junit jars
- Adding reference to the Junit jar in your project.
- Defining and writing our application under test.
- Writing our first simple JUnit test.
- *Choice of development environment is Eclipse.*
- Step 1: Downloading JUnit Jars
- To use **Junit** we need to have '**hamcrest-core-1.3-sources.jar**'& '**junit-4.10.jar**' Jar files included in the java project. You can download these Jars from '**https://github.com/junit-team/junit/wiki/Download-and-Install**'.
- Step 2: Setting up your project in Eclipse IDE

I hope you have gone through our tutorials in **installing** and **setting up Eclipse** here. Create a project in eclipse and name it '**JUnit Project**

Step 3: Add Junit Jar files

Well, there is nothing like installing *Junit*. All you have to do is to reference your *Junit jars* to your project. This can be done following these steps:

Right click your project in Package explorer and Go to properties.

Select Java Build path from the left pane.

Click on Add external Jars in the right pane.

Browse to the path where you have downloaded the jar files and include both the jars.

What is our application under test?

To perform *unit testing* you need to have an application (*Development code you may say*). Lets first define a simple application which will be our ***Application under test*** (*AUT*). Lets say that we have a small Math class which provides us with following *methods:*

- *- Add*
  – Multiply
  – Subtract

These *methods* will do exactly as they say. They will *add*, *multiply* and *subtract* two number. Now this*Math* class is our *AUT*. Here is my simple implementation of this class.

```
package Application;
/*
 * This is our sample class that we would like to unit test
 * To do this we will create a sample functionality in the
 * class
 * Let the class give us these 3 basic functionalities
 * Substract()
 * Add()
 * Multiply()
```

```
 *
 *  Our goal is to have unit tests around these
 *  basic functionalities
 */
public class MathProvider {
public MathProvider(){}
public int Substract(int firstNumber, int secondNumber)
{
return (firstNumber - secondNumber);
public int Add(int firstNumber, int secondNumber)
return (firstNumber + secondNumber);
}
public int Multiply(int firstNumber, int secondNumber)
{
return (firstNumber * secondNumber);
} }
```

- First Junit Test

Now that we have our *AUT* class present we will write our first test. In *Junit* we write tests in the form of*classes*. Each class has a *method* or a *set of methods* which test a particular functionality of the Unit. Here we will first write a simple test to verify that *MathProvider* class is able to give us right addition result.

```
package Application.UnitTests;
import static org.junit.Assert.*;
import junit.extensions.*;
import org.junit.Test;
import Application.MathProvider;
public class JunitMathProvider_1 {
MathProvider provider;
public SimpleTest(){
provider = new MathProvider();
@Test
public void add()
{
System.out.println("Starting test " + new Object(){}.getClass().getEnclosingMethod().getName());
   int firstNumber = 10;
   int secondNumber = 20;
assertEquals(firstNumber + secondNumber, provider.Add(firstNumber, secondNumber));
System.out.println("Ending test " + new Object(){}.getClass().getEnclosingMethod().getName());
} } }
```

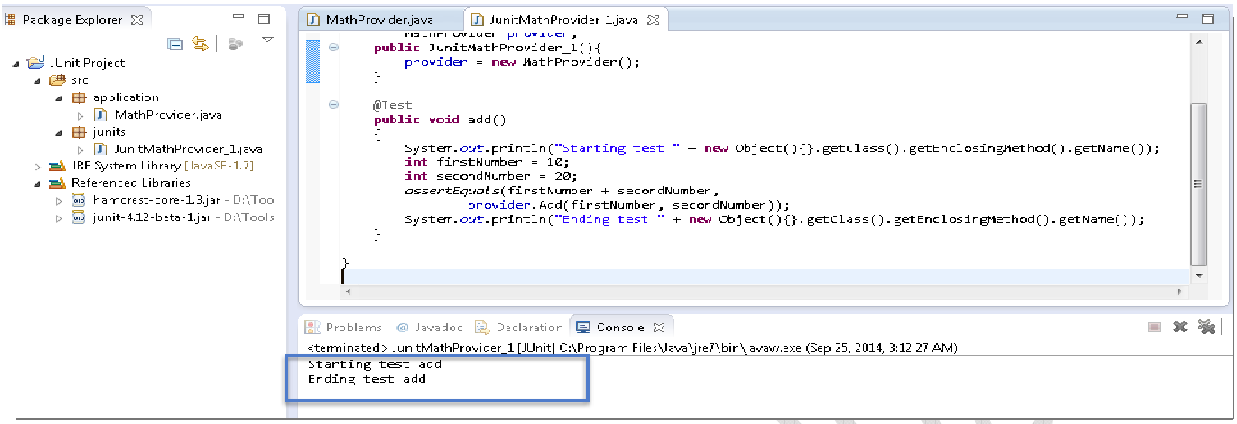
What is Asserts?

Asserts mean verifying the values. Assert can be of Object type, Boolean type, Int type or any data type. It simply verify the actual value with the expected value and asserts records only the failed scenario.

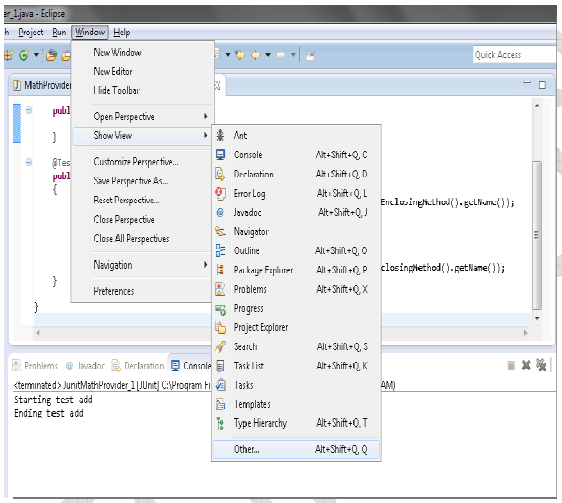
What is *'@Test'* Annotation?

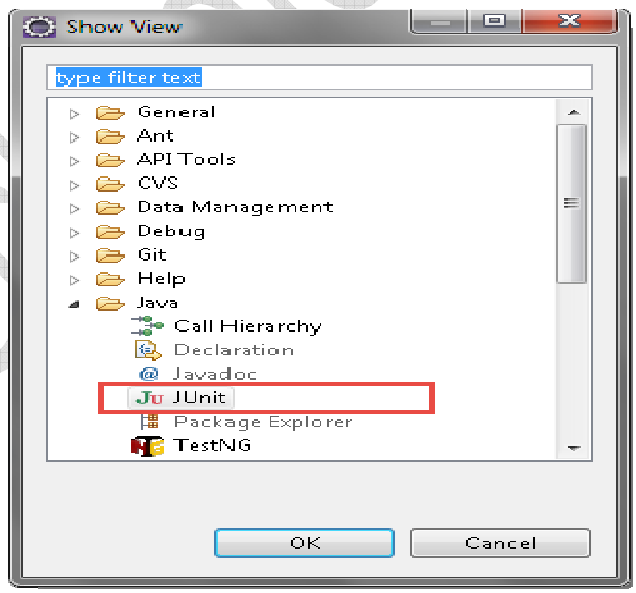
Pay attention to the *'@Test'* annotation. All the tests in *Junit4* has a *@Test* annotation that should be put on a test method. This helps the *Junit framework* to identify tests inside a test class.

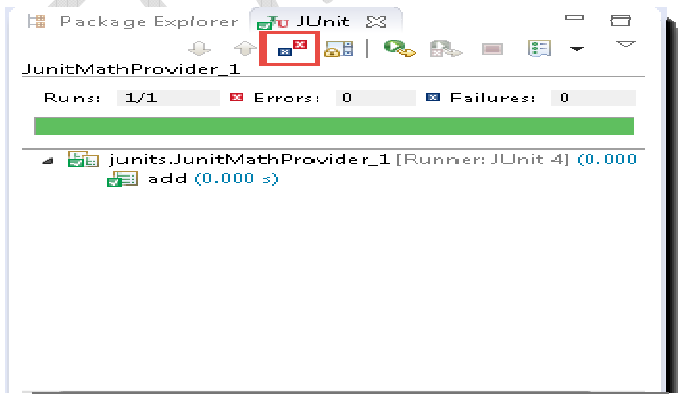
Run the First JUnit Test



- Set up JUnit Perspective:



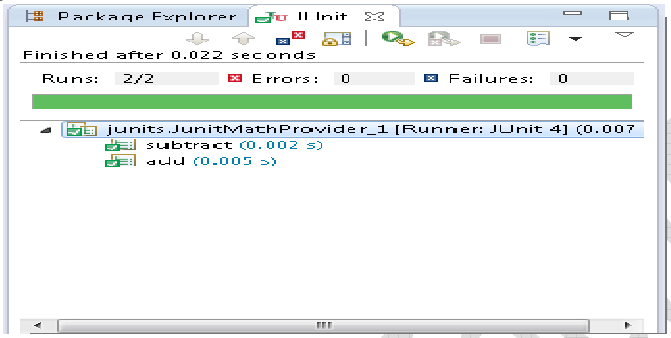




Multiple Tests Method and Test Classes

Moving ahead we would like to add more tests in our previously created test lass **JunitMathProvider_1**. Lets do that. We already have a test method which tests the **MathProvider.Add()** method. Lets add another method to test **MathProvider.Subtract()**. Here is how that subtract method will look like:

```
@Test
        public void subtract()
        {
                System.out.println("Starting test " + new
Object(){}.getClass().getEnclosingMethod().getName());
int firstNumber = 10;
 int secondNumber = 20;
assertEquals(firstNumber - secondNumber, provider.Substract(firstNumber, secondNumber));
System.out.println("Ending test " + new Object(){}.getClass().getEnclosingMethod().getName());
}
```



Adding extra Junit Test Class

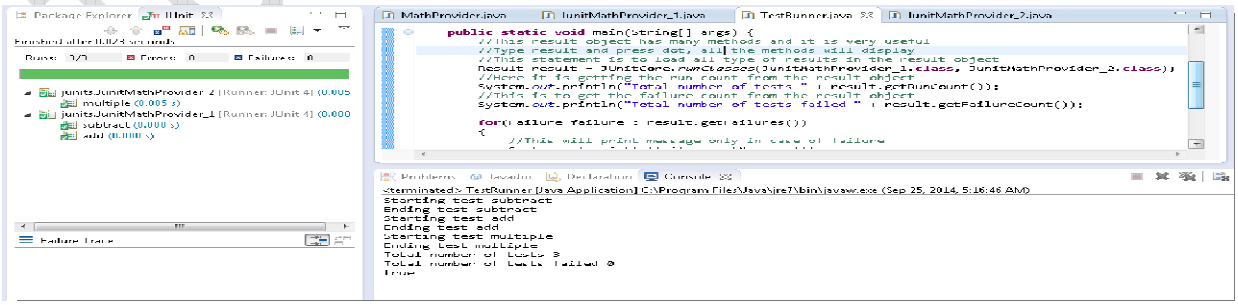
Lets start writing another test class in our *Junit* and lets name it '***JunitMathProvider_2'***. The primary focus of this test class should be to test *MathProvider.Multiply method*. This class is some what similar to the previous class but has a test method named *multiply()* instead of *add()* and *subtract()* test methods. Class will look like this

```
package Application.UnitTests;
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;
package Application.UnitTests;
import static org.junit.Assert.assertEquals;
import org.junit.Test;
import Application.MathProvider;
public class JunitMathProvider_2 {
MathProvider provider;
public JunitMathProvider_2{
provider = new MathProvider();
};
```

```
@Test
public void multiply()
{
System.out.println("Starting test " + new Object(){}.getClass().getEnclosingMethod().getName());
   int firstNumber = 10;
   int secondNumber = 20;
assertEquals(firstNumber * secondNumber, provider.Multiply(firstNumber, secondNumber));
System.out.println("Ending test " + new Object(){}.getClass().getEnclosingMethod().getName());
} }
```

Running Test Methods and Test Classes

```
package junits;
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;
public class TestRunner {
public static void main(String[] args) {
//This result object has many methods and it is very useful
//Type result and press dot, all the methods will display
//This statement is to load all type of results in the result object
Result result = JUnitCore.runClasses(JunitMathProvider_1.class, JunitMathProvider_2.class);
//Here it is getting the run count from the result object
System.out.println("Total number of tests " + result.getRunCount());
//This is to get the failure count from the result object
System.out.println("Total number of tests failed " + result.getFailureCount());
for(Failure failure : result.getFailures())
{
//This will print message only in case of failure
System.out.println(failure.getMessage());    }
//This will print the overall test result in boolean type
System.out.println(result.wasSuccessful());
} }
```



Junit Test with Selenium WebDriver

```
package unitTests;
import java.util.concurrent.TimeUnit;
import org.junit.AfterClass;
import org.junit.Assert;
import org.junit.BeforeClass;
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
public class SeleniumTest {
 private static FirefoxDriver driver;
 WebElement element;
 @BeforeClass
    public static void openBrowser(){
       driver = new FirefoxDriver();
       driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
}
@Test
    public void valid_UserCredential(){
 System.out.println("Starting test " + new Object(){}.getClass().getEnclosingMethod().getName());
    driver.get("http://www.store.demoqa.com");
    driver.findElement(By.xpath(".//*[@id='account']/a")).click();
    driver.findElement(By.id("log")).sendKeys("testuser_3");
    driver.findElement(By.id("pwd")).sendKeys("Test@123");
    driver.findElement(By.id("login")).click();
    try{
element = driver.findElement (By.xpath(".//*[@id='account_logout']/a"));
 }catch (Exception e){
}
 Assert.assertNotNull(element);
  System.out.println("Ending test " + new Object(){}.getClass().getEnclosingMethod().getName());
   }
 @Test
 public void inValid_UserCredential()
    {
 System.out.println("Starting test " + new Object(){}.getClass().getEnclosingMethod().getName());
 driver.get("http://www.store.demoqa.com");
 driver.findElement(By.xpath(".//*[@id='account']/a")).click();
  driver.findElement(By.id("log")).sendKeys("testuser");
  driver.findElement(By.id("pwd")).sendKeys("Test@123");
try{
```

```
element = driver.findElement (By.xpath(".//*[@id='account_logout']/a"));
    }catch (Exception e){
}
 Assert.assertNotNull(element);
    System.out.println("Ending test " + new Object(){}.getClass().getEnclosingMethod().getName());
    }
 @AfterClass
 public static void closeBrowser(){
 driver.quit();
 }
}
driver.findElement(By.id("login")).click();
```

Parameterization in JUnit

Parameterized class is used to run same scenario with multiple dataset.

Below is the example to pass multiple parameters in a Junit test.

@Parameters annotation tag is used to pass multiple data. Here, we have taken 2*2 dimensional array and the data can be visualized like below:

```
@RunWith(Parameterized.class)
public class Junttest {
public String name;
public int age;
public Junttest(String name,int age){
this.name=name;
this.age=age; }
@Test
public void testMethod(){
System.out.println("Name is: "+name +" and age is: "+age);
}
@Parameters
public static Collection<Object[]> parameter(){
Object[][] pData=new Object[2][2];
pData[0][0]="Tom";
pData[0][1]=30;
pData[1][0]="Harry";
pData[1][1]=40;
return Arrays.asList(pData);  }  }
```

JUnit Assertions

JUnit assertEquals: This checks if the two values are equal and assertion fails if both values are not equal.

This compares Boolean, int, String, float, long, char etc.

Syntax:

Assert.assertEqual("excepted value", "actual value");

Example:
Assert.assertEqual("ABC","ABC"); //Both the strings are equal and assertion will pass.
Assert.assertEqual("ABC","DEF"); //Assertion will fail as both the strings are not equal.
JUnit assertTrue: Returns true if the condition is true and assertion fails if the condition is false.
Assert.assertTrue("message", condition);
Assert.assertTrue("Both the strings are not equal", ("HelloWorld").equals("HelloWorld"));
JUnit assertFalse: Returns true if the condition is false and assertion fails if the condition is true.
Assert.assertFalse("message", condition);
Assert.assertFalse("Both the strings are equal", ("Hello").equals("HelloWorld"));
Skip the test
In JUnit, @Ignore annotation is used to skip or ignore a test case/ test method if it is not ready to test. We need to import a statement "import org.junit.Ignore;" when working with @Ignore.
@Ignore("Im Not Ready") @Test **public void testCaseAccounting**(){ System.out.println("I'm not ready, please don't execute me"); }
Report generation in Junit
JUnit is one of those unit frameworks which were initially used by many Java applications as a Unit test framework. By default, JUnit tests generate simple report XML files for its test execution.
These XML files can then be used to generate any custom reports as per the testing requirement. We can also generate HTML reports using the XML files.
Ant has such a utility task, which takes these JUnit XML files as input and generates an HTML report.
TestNG, by default, generates JUnit XML reports for any test execution (in the *test-output*folder). We can use these XML report files as input for generating a JUnit HTML report. Let us take an example.
Create Test Case Class

```
import org.testng.Assert; import org.testng.annotations.Test;
 public class SampleTest {
@Test
 public void testMethodOne(){
Assert.assertTrue(true);
}
@Test public void testMethodTwo()
{
Assert.assertTrue(false);
}
@Test(dependsOnMethods={"testMethodTwo"})
public void testMethodThree(){
Assert.assertTrue(true);
 } }
```

The preceding test class contains three test methods out of
which *testMethodOne* and*testMethodThree* will pass when executed, whereas *testMethodTwo* is made

to fail by passing a *false* Boolean value to the Assert.assertTrue method, which is used for truth conditions in the tests.

Create testng.xml

Create testng.xml in **C:\ > TestNG_WORKSPACE** to execute test case(s).

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="Simple Suite">
 <test name="Simple test">
<classes>
<class name="SampleTest" /> </classes>
</test> </suite>
```

Compile the SampleTest class using javac.

```
C:\TestNG_WORKSPACE>javac SampleTest.java
```

Now, run testng.xml.

```
C:\TestNG_WORKSPACE>java -cp "C:\TestNG_WORKSPACE" org.testng.TestNG testng.xml
```

- **Verify the output.**

Simple Suite Total tests run: 3, Failures: 1, Skips: 1

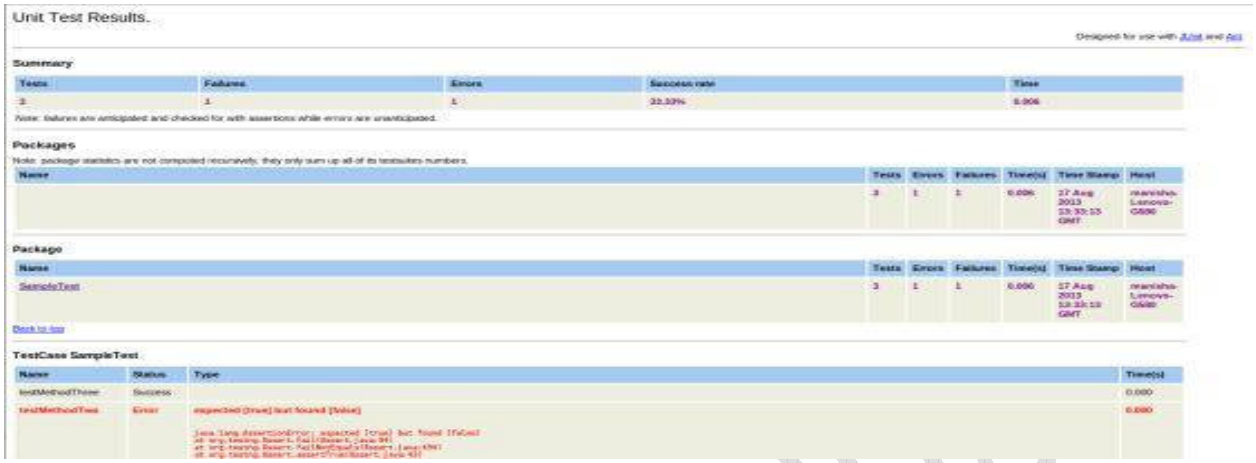
Now that we have JUnit XML reports available from the above execution, let's create a simple Ant build configuration XML file to generate an HTML report for the test execution.

Create a new file named build.xml under **C:\ > TestNG_WORKSPACE** folder.

```
<project name="TestNG_WORKSPACE" default="junit-report" basedir=".">
<!-- Sets the property variables to point to respective directories --> <property name="junit-xml-dir"
value="${basedir}/test-output/junitreports"/>
 <property name="report-dir" value="${basedir}/html-report" /> <!-- Ant target to generate html report
-->
 <target name="junit-report">
<!-- Delete and recreate the html report directories -->
<delete dir="${report-dir}" failonerror="false"/>
<mkdir dir="${report-dir}" />
 <mkdir dir="${report-dir}/Junit" />
 <!-- Ant task to generate the html report. todir - Directory to generate the output reports fileset -
Directory to look for the junit xml reports. report - defines the type of format to be generated. Here we
are using "noframes" which generates a single html report. -->
 <junitreport todir="${report-dir}/Junit">
 <fileset dir="${junit-xml-dir}">
 <include name="**/*.xml" />
 </fileset>
<report format="noframes" todir="${report-dir}/Junit" />
 </junitreport> </target> </project>
```

Open the command prompt window and go to the C:\ > TestNG_WORKSPACE directory in the command prompt and run the command:

C:\TestNG_WORKSPACE> antOnce executed, a JUnit HTML report will be generated in the configured directory /html-report/Junit. Open the file named junit-noframes.html on your default web browser. You will see the following HTML report:

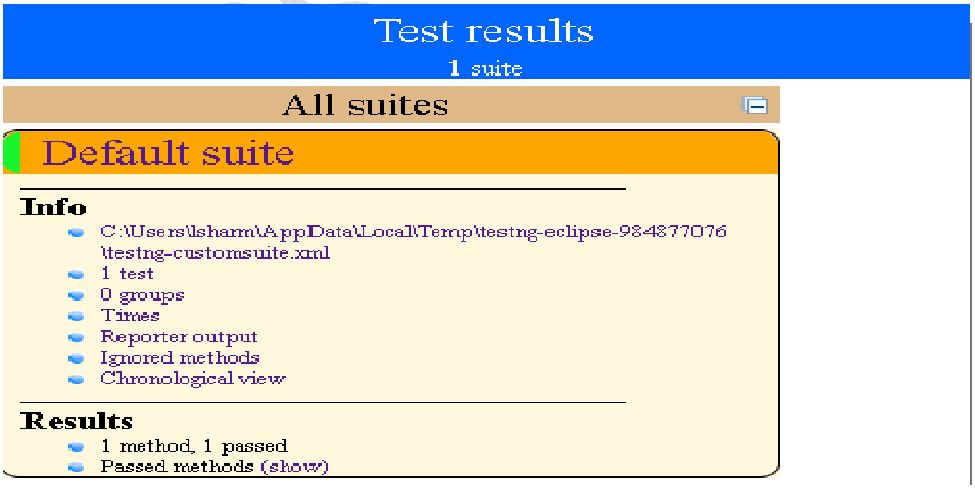


TestNG Reporting

TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use. In simple words TestNG is a tool that help us to organize the tests and help us to produce the test reports. TestNG framework can be used for automation testing with Selenium (web application automation testing tool).

TestNG Advantages

- Multiple built in Annotations which are easier to use and understand
- Test method can be dependent to other method
- Test cases can be Grouped and can be execute separately by groups
- Parallel testing is possible
- TestNG has built in HTML report and XML report generation facility. It has also built in logging facility



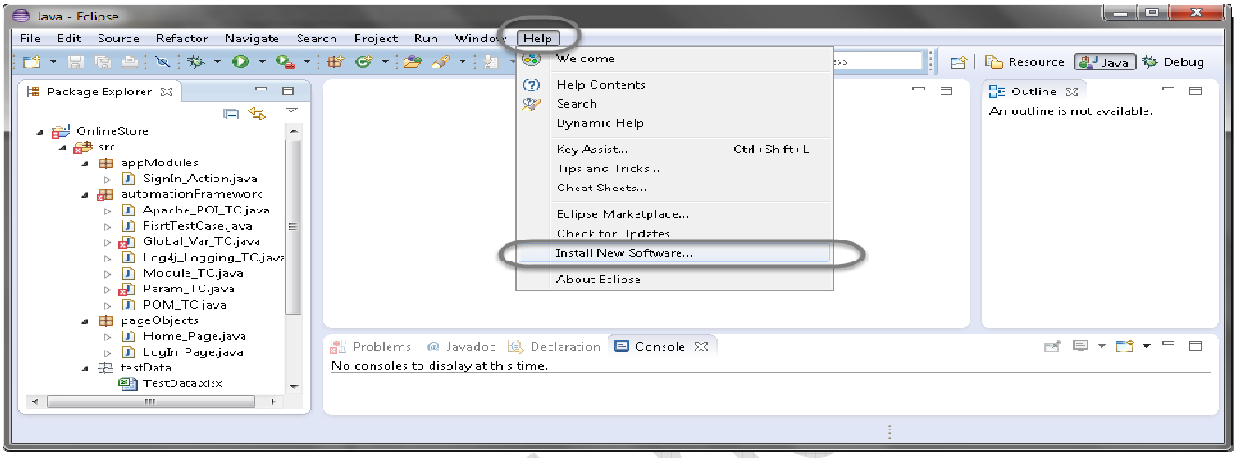
Integration of eclipse and TestNG

Install TestNG in Eclipse

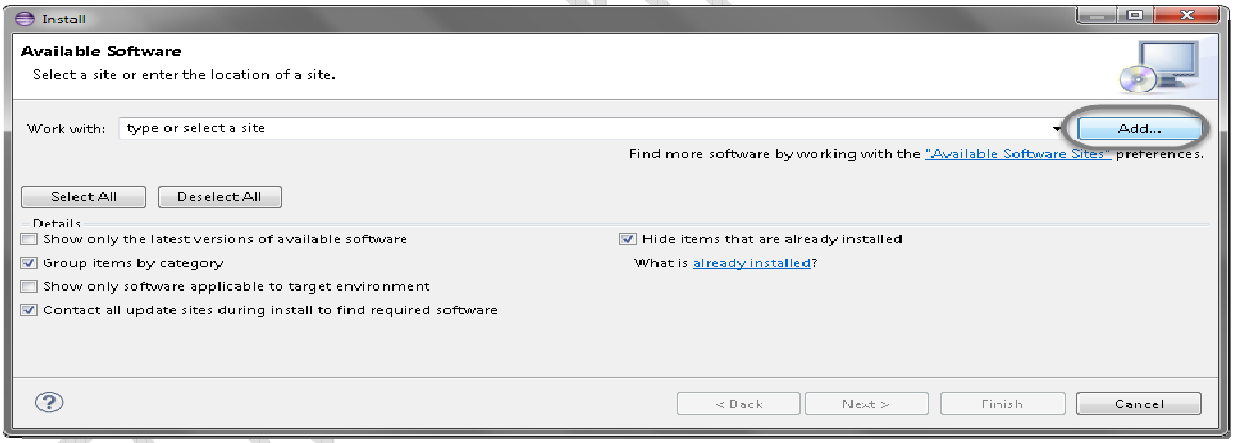
It is easy to install TestNG, as it comes as a plugin for Eclipse IDE. Prerequisite for installing TestNG is your Internet connection should be up & running during installation of this plugin and Eclipse IDE should be installed in your computer. Please see Download and Install Eclipse to setup Eclipse to you system.

Steps to follow:

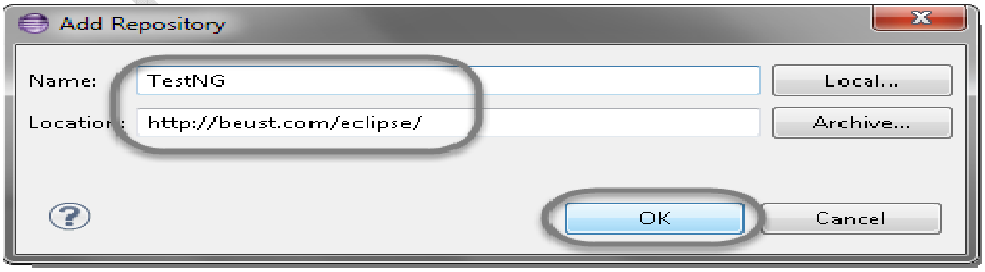
1) Launch the Eclipse IDE and from Help menu, click "**Install New Software**".



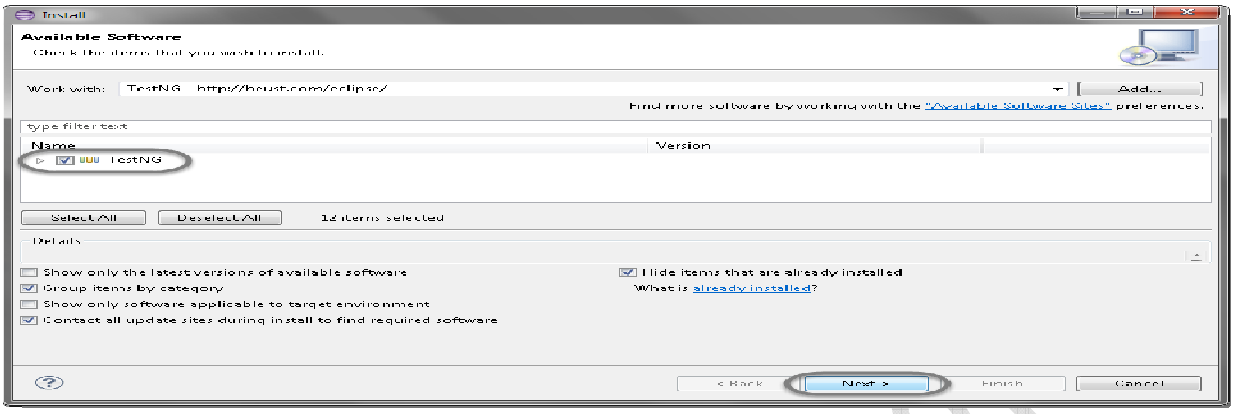
2) You will see a dialog window, click "**Add**" button.



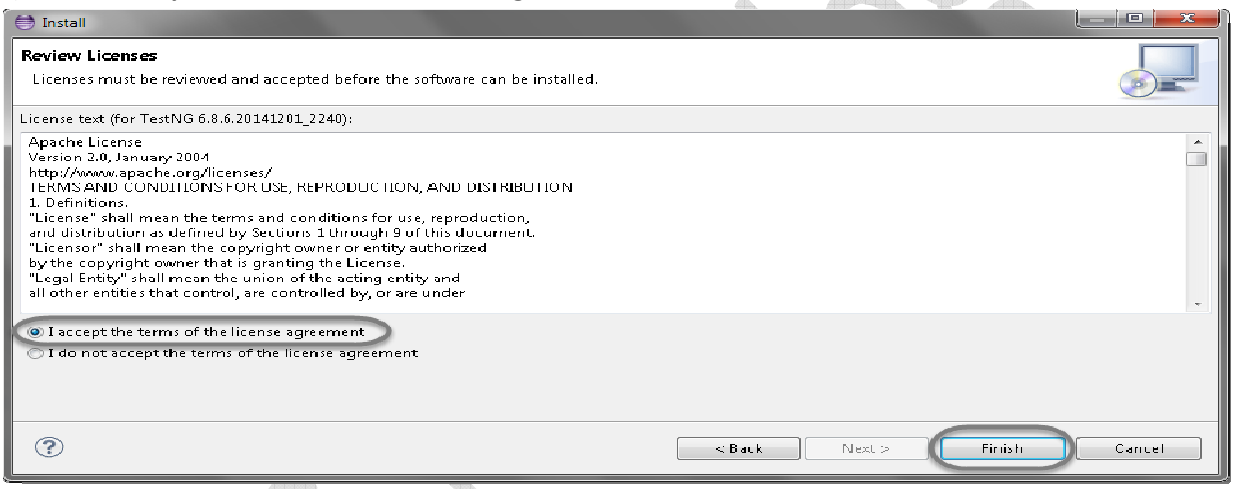
3) Type name as you wish, lets take "**TestNG**" and type "**http://beust.com/eclipse/**" as location. Click OK.



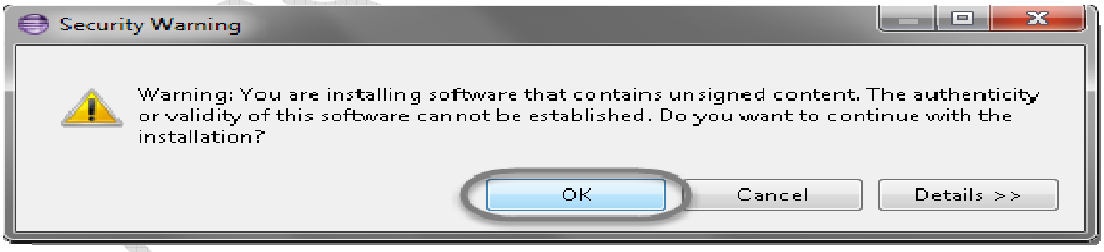
4) You come back to the previous window but this time you must see TestNG option in the available software list. Just Click TestNG and press "**Next**" button.



5) Click "**I accept the terms of the license agreement**" then click **Finish**.

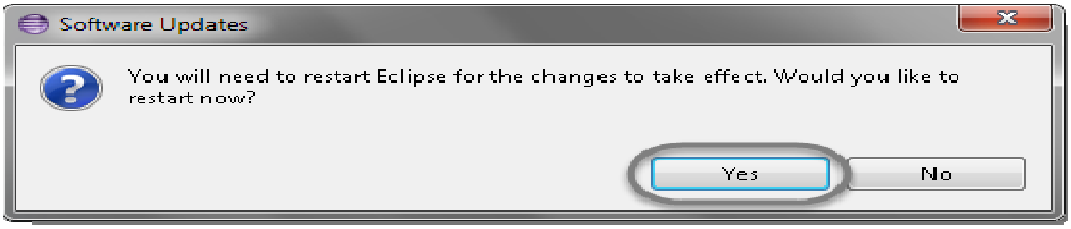


- 6) You may or may not encounter a Security warning, if in case you do just click **OK**.



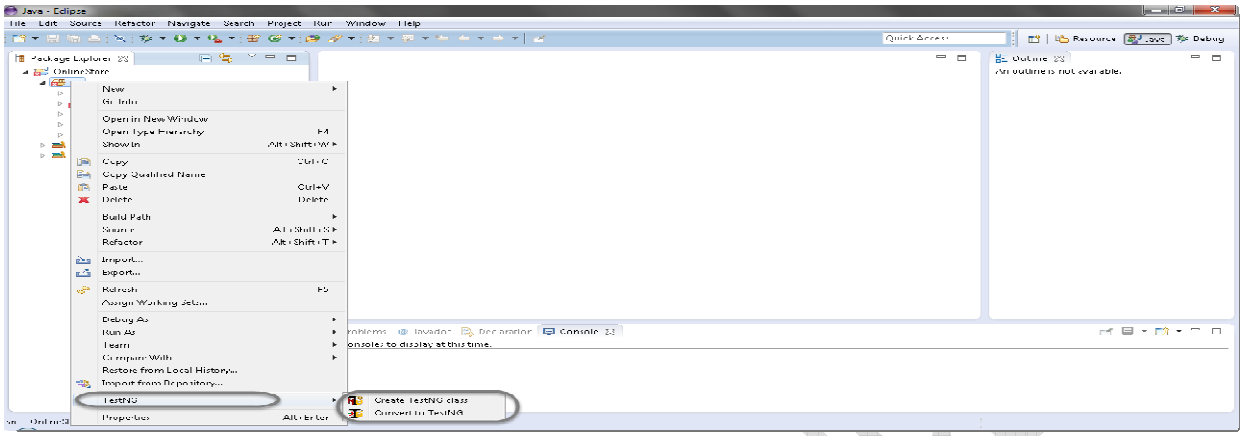
7) Click **Next** again on the succeeding dialog box until it prompts you to Restart the Eclipse.

8) You are all done now, just Click **Yes**.

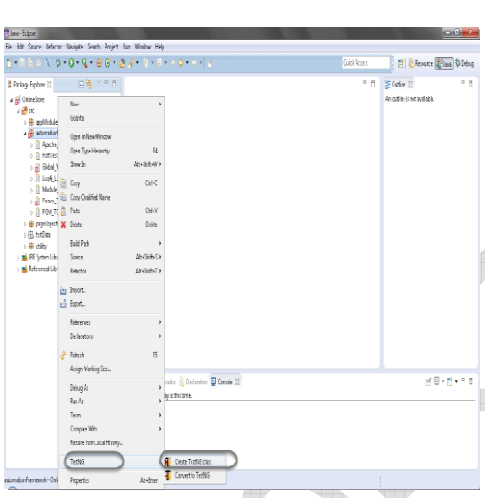


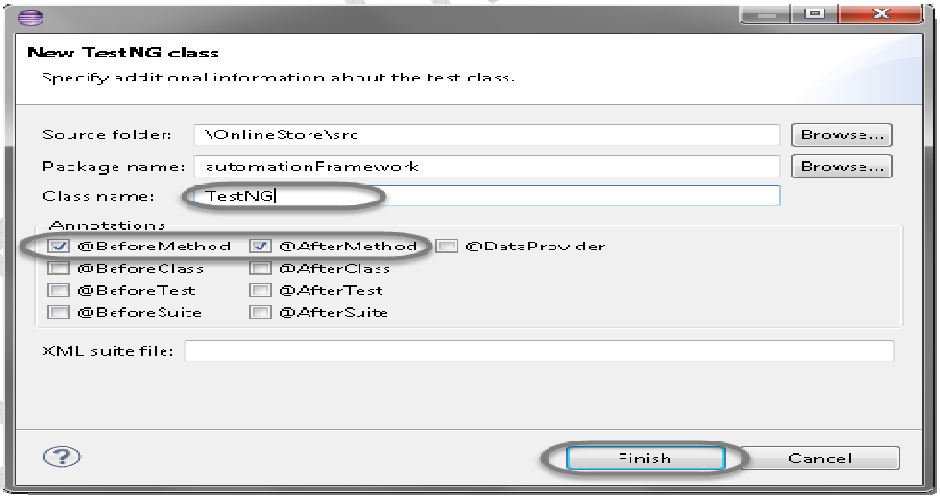
9) Proceed with your workplace.

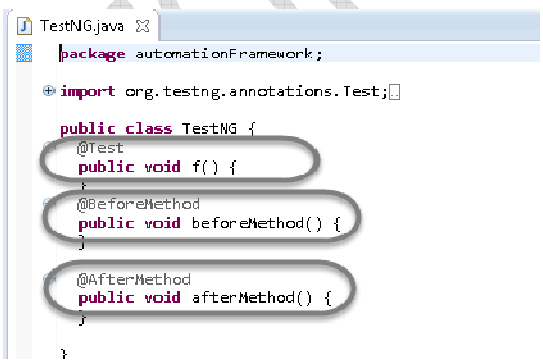
10) After restart, verify if TestNG was indeed successfully installed. Right click on you project and see if**TestNG** is displayed in the opened menu.

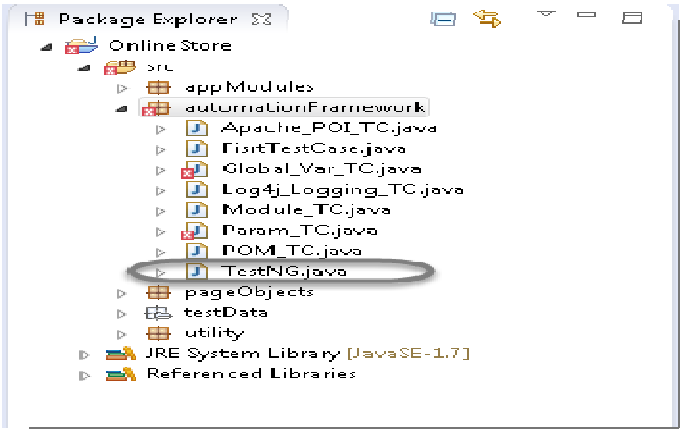


- Create TestNG class







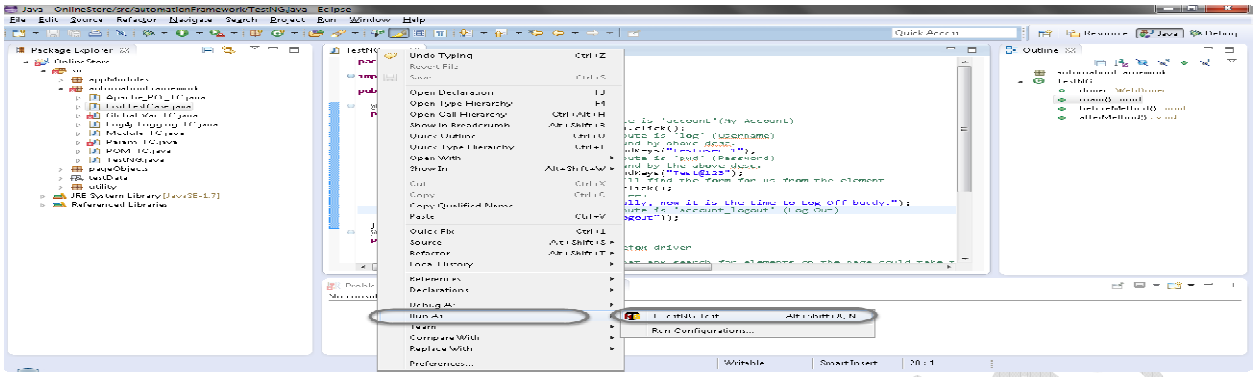


- Let's take an example of **First Test Case** and divide the test case in to three parts .
- **@BeforeMethod** : Launch Firefox and direct it to the Base URL
- **@Test** : Enter Username & Password to Login, Print console message and Log out

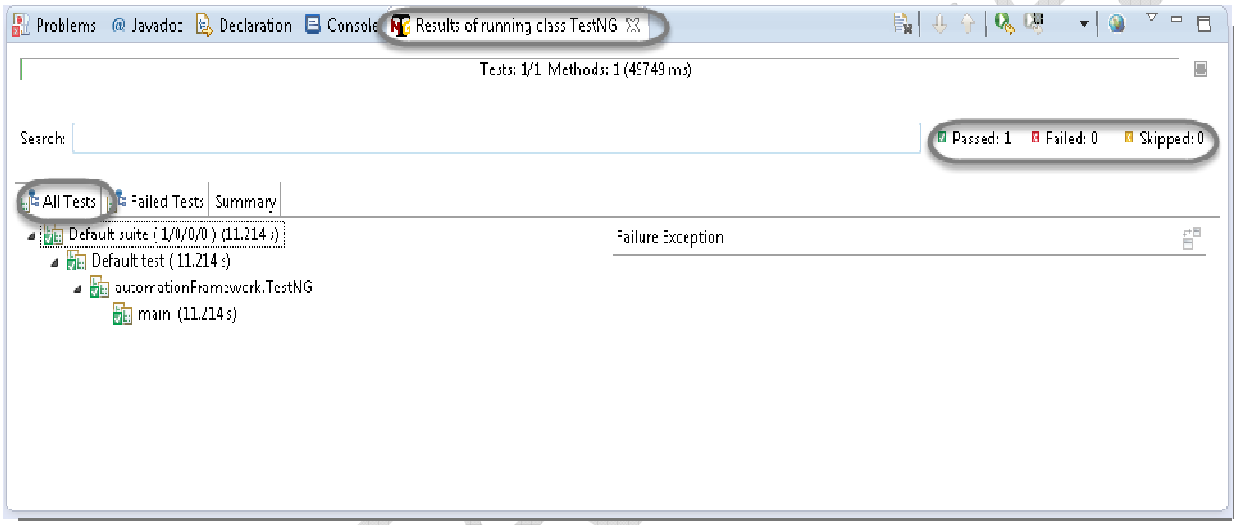
@AfterMethod : Close Firefox browser

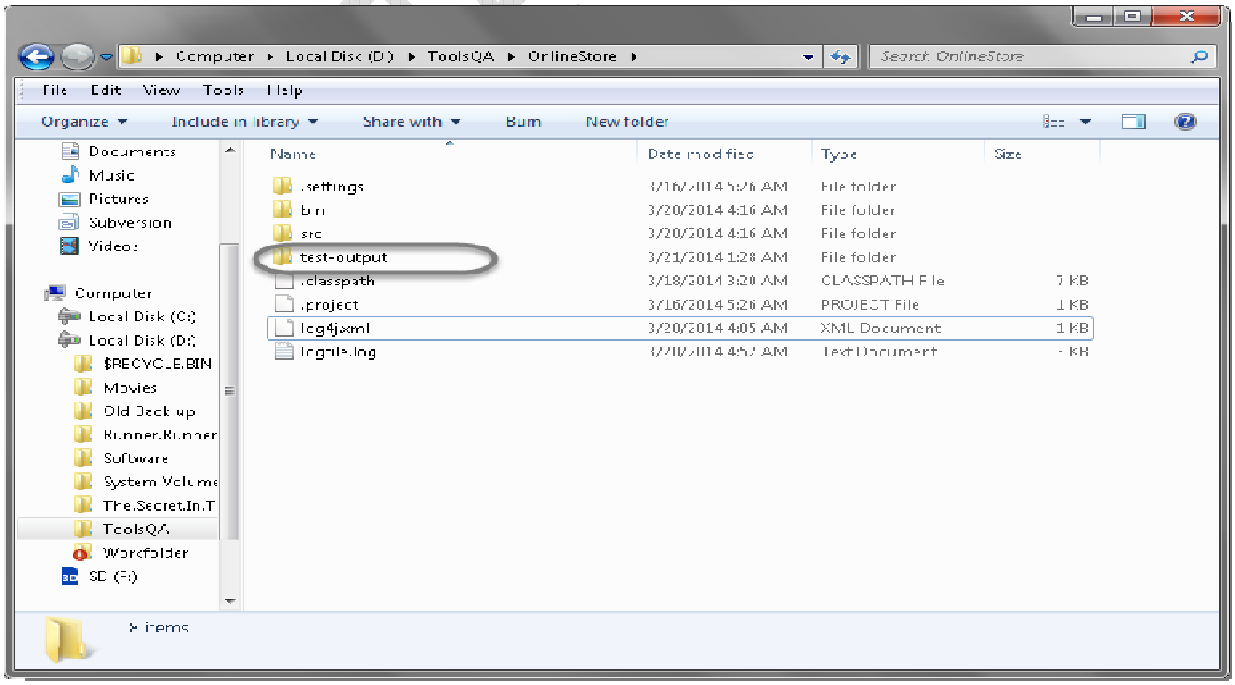
```
public class TestNG {
public WebDriver driver;
  @Test
  public void main() {
// Find the element that's ID attribute is 'account'(My Account)
     driver.findElement(By.id("account")).click();
     // Find the element that's ID attribute is 'log' (Username)
     // Enter Username on the element found by above desc.
     driver.findElement(By.id("log")).sendKeys("testuser_1") // Find the element that's ID attribute is
'pwd' (Password)

    // Enter Password on the element found by the above desc.
      driver.findElement(By.id("pwd")).sendKeys("Test@123");
      // Now submit the form. WebDriver will find the form for us from the element
      driver.findElement(By.id("login")).click();
      // Print a Log In message to the screen
      System.out.println(" Login Successfully, now it is the time to Log Off buddy.");
      // Find the element that's ID attribute is 'account_logout' (Log Out)
     driver.findElement(By.id("account_logout"));
  }
@BeforeMethod
   public void beforeMethod() {
   // Create a new instance of the Firefox driver
      driver = new FirefoxDriver();
      //Put a Implicit wait, this means that any search for elements on the page could take the time the
implicit wait is set for before throwing exception
      driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
      //Launch the Online Store Website
 driver.get("http://www.onlinestore.toolsqa.com");
   }
   @AfterMethod
   public void afterMethod() {
   // Close the driver
      driver.quit();
   }
}
```

Run the test by right click on the test case script and select **Run As** > **TestNG Test**.



Results of running the Testng Test Case





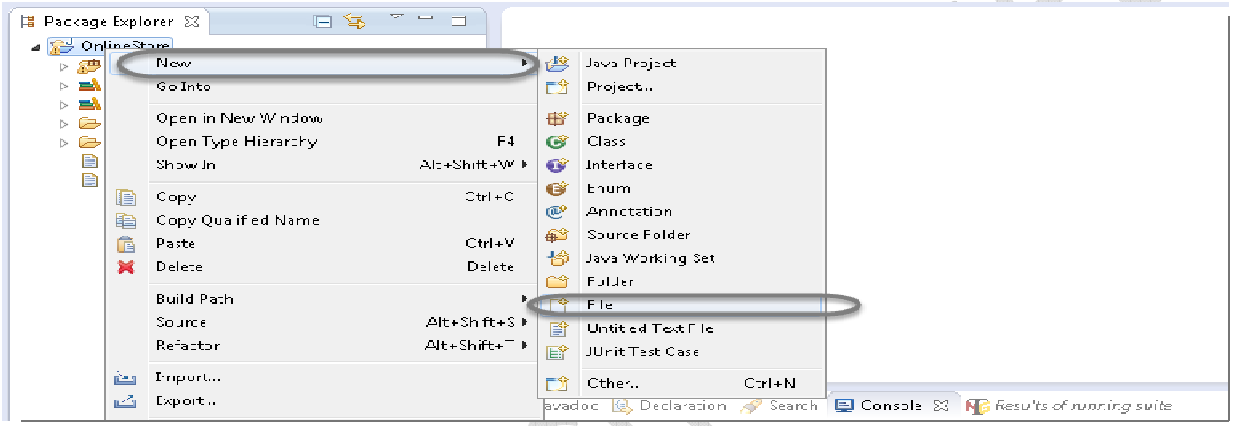
In any project, you will end up to a place where you need to execute so many test cases on a run. Running a set of test cases together is call executing a **Test Suite**. Those test cases can be dependent to each other or may have to be executed in a certain order. TestNg gives us the capability to manage our test execution

In TestNG framework, we need to create **testng.xml** file to create and handle multiple test classes. This is the xml file where you will configure your test run, set test dependency, include or exclude any test, method, class or package and set priority etc.

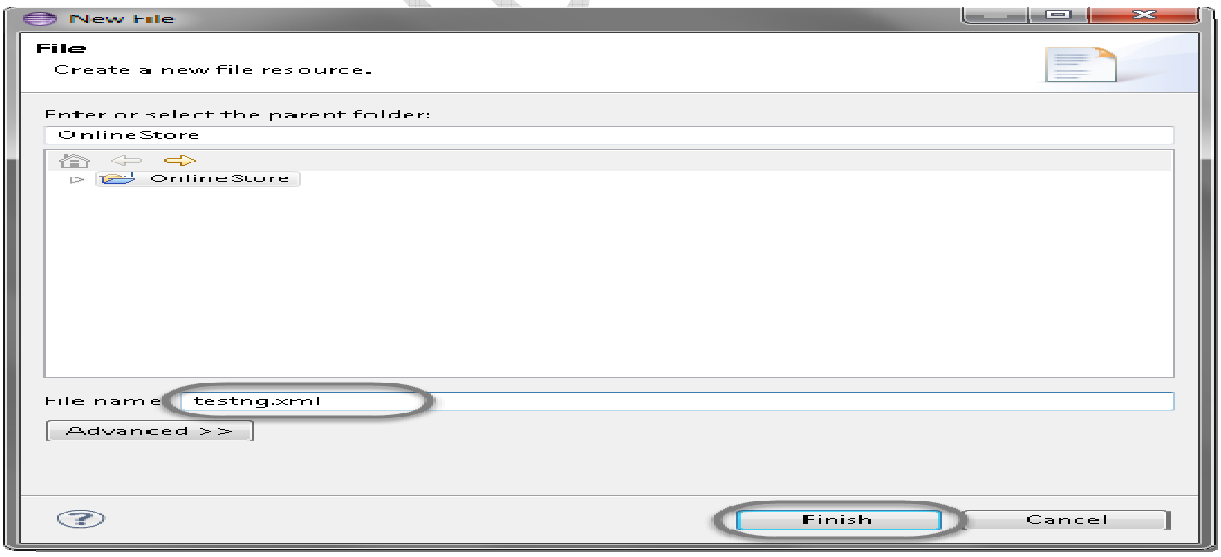
How to do it…

Step 1 : Create a TestNG XML

1) Right click on Project folder, go to **New** and select '**File**' as shown in below image.



2) In New file wizard, add file name = '**testng.xml**' as shown in below given image and click on **Finish**button.



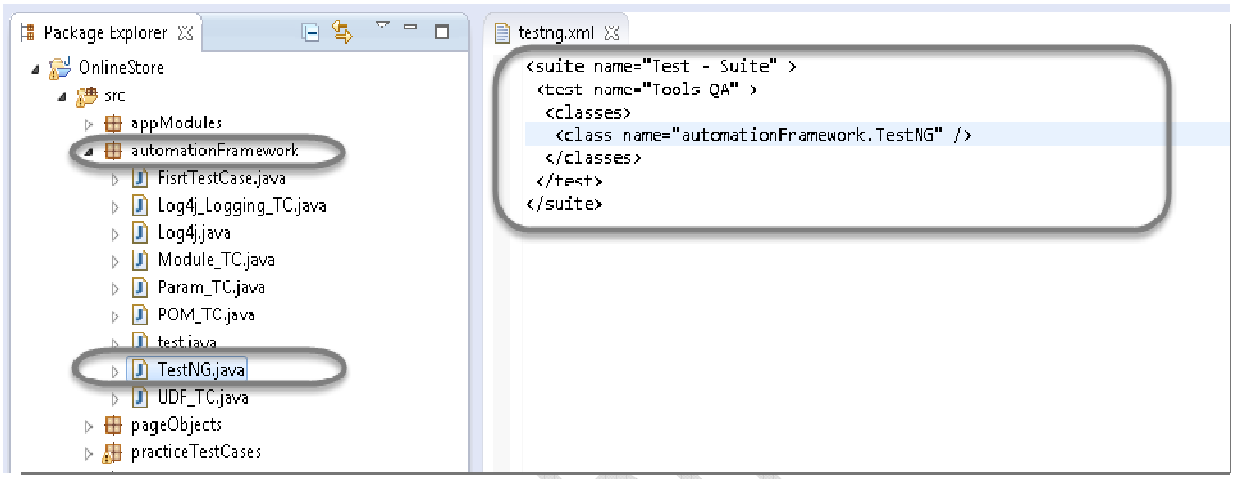
3) It will add **testng.xml** file under your project folder.

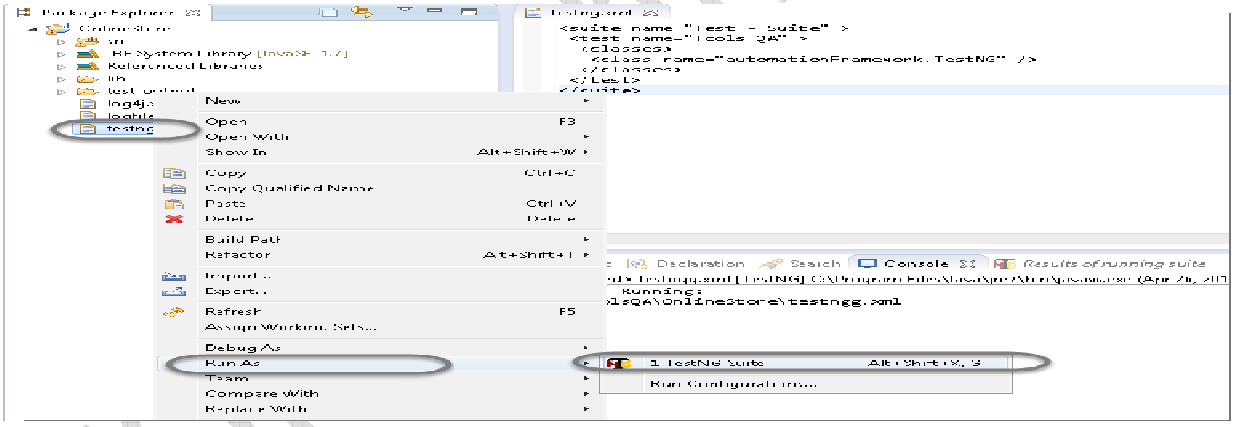
Step 2 : Write xml code ?

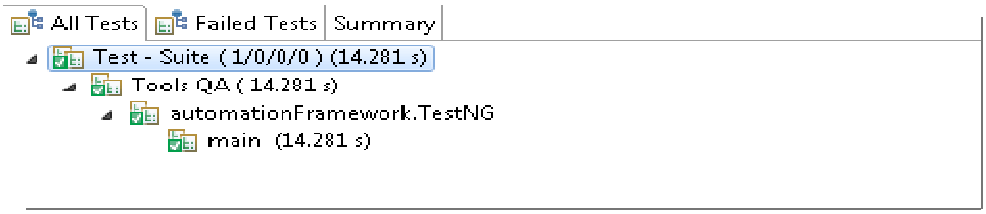
1) Now add below given code in your testng.xml file.

```
<suite name="Any Name" >
        <test name="Any Name“>
         <classes><class name="PackageName.TestCaseName" />
        <c/lasses>
        </test>
</suite>
```

2) After giving appropriate names, now your testng.xml file will looks like this:

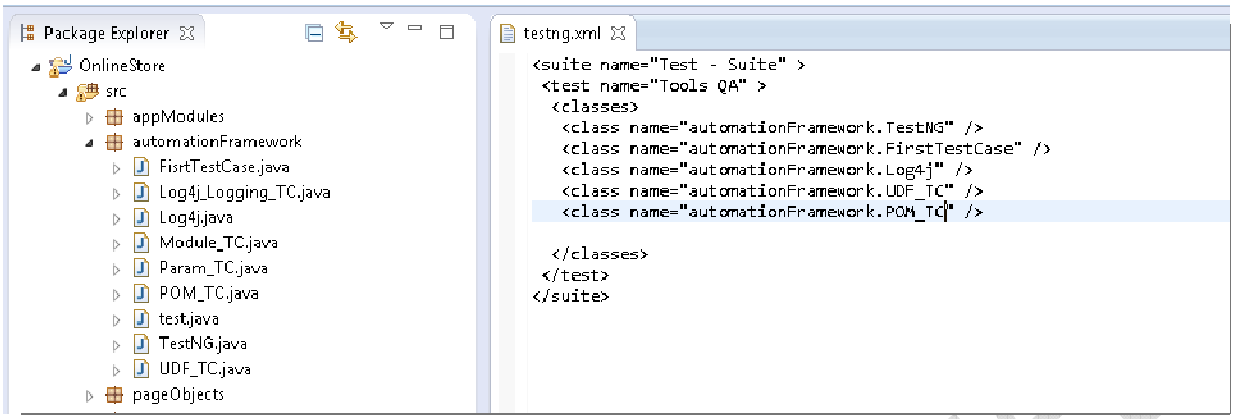






Building a Test Suite

Now when you have learned how to build the xml, now it’s time to learn how to build a Test Suite using testng.xml. It is again not a complex task, all you need to do is to add your test cases to your xml file in <classes> tag.



TestNG Annotations

In the TestNG Introduction chapter we have came across different annotations used in TestNG Framework but so far we have used just three(Before, After & Test). All though these are the most frequently used annotations but who know how far you will go with your framework and may like to use other useful TestNG annotations.

Before that I would like you to give a small idea on Annotations hierarchy or Annotations levels in TestNG.

```
<suite>
        <test>
            <classes>
                <method>
                    <test>
                </method>
            </classes>
        </test>
</suite>
```

```
package automationFramework;
import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.AfterSuite;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;
public class Sequencing {
@Test
public void testCase1() {
System.out.println("This is the Test Case 1");
```

```
}
@Test
public void testCase2() {
System.out.println("This is the Test Case 2");
}
@BeforeMethod
public void beforeMethod() {
System.out.println("This will execute before every Method");
}
@AfterMethod
public void afterMethod() {
System.out.println("This will execute after every Method");
}
@BeforeClass
public void beforeClass() {
System.out.println("This will execute before the Class");
}
@AfterClass
public void afterClass() {
System.out.println("This will execute after the Class");
                                }
@BeforeTest
public void beforeTest() {
System.out.println("This will execute before the Test");
}
@AfterTest
public void afterTest() {
System.out.println("This will execute after the Test");
}
@BeforeSuite
public void beforeSuite() {
System.out.println("This will execute before the Test Suite");
}
@AfterSuite
public void afterSuite() {
System.out.println("This will execute after the Test Suite");
} }
```

Data Driven Testing

A key benefit of automating functional testing is the ability to test large volumes of data on the system quickly. But you must be able to manipulate the data sets, perform calculations, and quickly create

hundreds of test iterations and permutations with minimal effort. Test Automation Frameworks must have capability to integrate with spreadsheets and provide powerful calculation features.

TestNG Data Providers

When you need to pass complex parameters or parameters that need to be created from Java (complex objects, objects read from a property file or a database, etc…), in such cases parameters can be passed using Dataproviders. A Data Provider is a method annotated with @DataProvider. A Data Provider returns an array of objects.

Let us check out the same Sign In examples using Data Providers with Excel data sheet.

How to do it…

Here we will follow a simple step by step process to Implement Excel with TestNg Data Provider.

Step 1: Create a test case of Login Application with TestNG Data Provider.

Step 2: Create a Test Data sheet.

Step 3: Create functions to Open & Read data from Excel

Step 4: Create a TestNg test case for accepting data from Excel using Data Provider.

Step 5: Run the test against the Test Case name in the Test Data file.

Step 1: Create a test case of LogIn Application with TestNG Data Provider

```
package automationFramework;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;
public class DataProviderTest {
   private static WebDriver driver;
  @DataProvider(name = "Authentication")
  public static Object[][] credentials() {
      // The number of times data is repeated, test will be executed the same no. of times
      // Here it will execute two times
      return new Object[][] { { "testuser_1", "Test@123" }, { "testuser_1", "Test@123" }};
  } // Here we are calling the Data Provider object with its Name

  @Test(dataProvider = "Authentication")
  public void test(String sUsername, String sPassword) {
     driver = new FirefoxDriver();
     driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
     driver.get("http://www.store.demoqa.com");
     driver.findElement(By.xpath(".//*[@id='account']/a")).click();
  // Argument passed will be used here as String Variable
     driver.findElement(By.id("log")).sendKeys(sUsername);
     driver.findElement(By.id("pwd")).sendKeys(sPassword);
```

```
            driver.findElement(By.id("login")).click();
            driver.findElement(By.xpath(".//*[@id='account_logout']/a")).click();
            driver.quit();
          }
        }
```

Step 2: Create a Test Data sheet

	A	B	C	D
1	**Test Case Name**	**UserName**	**Password**	
2	DataProviderWithExcel_001	testuser_1	Test@123	
3	DataProviderWithExcel_002	testuser_2	Test@223	
4				

Step 3: Create functions to Open & Read data from Excel

```
package utility;
 import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
public class ExcelUtils {
        private static XSSFSheet ExcelWSheet;
        private static XSSFWorkbook ExcelWBook;
        private static XSSFCell Cell;
        private static XSSFRow Row;
        public static Object[][] getTableArray(String FilePath, String SheetName) throws Exception {
String[][] tabArray = null;
 try {
  FileInputStream ExcelFile = new FileInputStream(FilePath);
        // Access the required test data sheet
         ExcelWBook = new XSSFWorkbook(ExcelFile);
          ExcelWSheet = ExcelWBook.getSheet(SheetName);
        int startRow = 1;
          int startCol = 1;
         int ci,cj;
          int totalRows = ExcelWSheet.getLastRowNum();
          // you can write a function as well to get Column count
          int totalCols = 2;
          tabArray=new String[totalRows][totalCols];
          ci=0;
          for (int i=startRow;i<=totalRows;i++, ci++) {
                  cj=0;
```

```
for (int j=startCol;j<=totalCols;j++, cj++){   tabArray[ci][cj]=getCellData(i,j);
System.out.println(tabArray[ci][cj]);                    }

                        }
}
        catch (FileNotFoundException e){
        System.out.println("Could not read the Excel sheet");
                        e.printStackTrace();
        }
        catch (IOException e){
System.out.println("Could not read the Excel sheet");
                        e.printStackTrace();
}
        return(tabArray);
}
public static String getCellData(int RowNum, int ColNum) throws Exception {
        try{
        Cell = ExcelWSheet.getRow(RowNum).getCell(ColNum);
                        int dataType = Cell.getCellType();
                        if  (dataType == 3) {
                        return "";
                        }else{
        String CellData = Cell.getStringCellValue();
                        return CellData;
        }catch (Exception e){
        System.out.println(e.getMessage());
        throw (e);
}       }       }
```

Maven

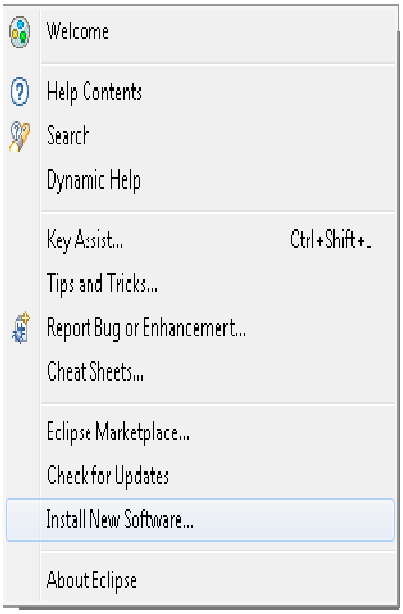
What is Maven?

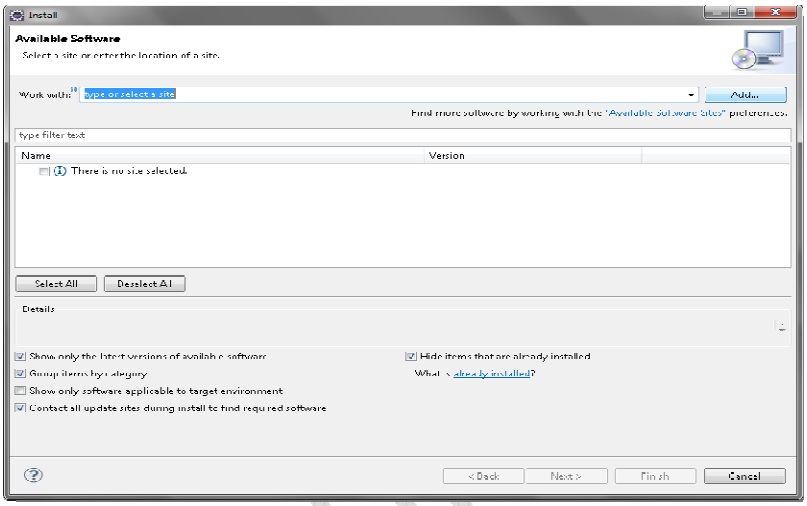
Maven is a build tool and it performs task just like Ant which is again a different build Tool. It is a software project management tool which provides new concept of project object model (POM). Maven allows the developer to automate the process of the creation of the initial folder structure, performing the compilation and testing and the packaging and deployment of the final product. It cuts down the good number of steps in build process and it makes it one step process to do a build.

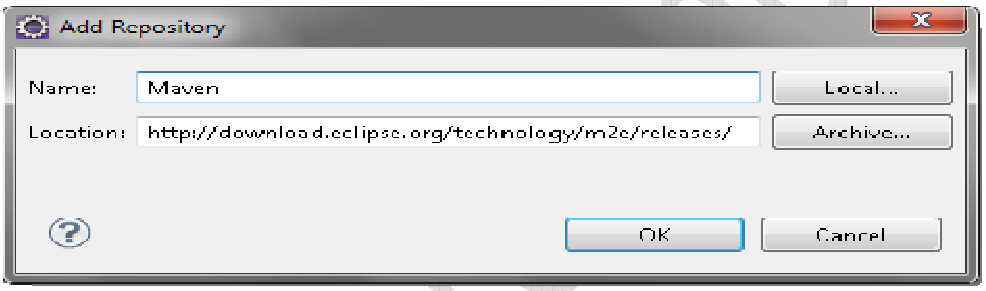
Why Maven is Used?

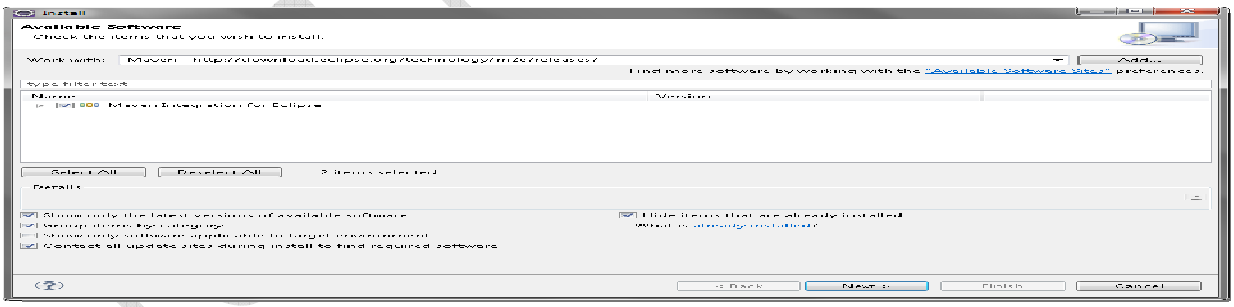
As I siad above it cuts down the tasks in build process. To summarize, Maven simplifies and standardizes the project build process. It handles compilation, distribution, documentation, team collaboration and other tasks seamlessly. Maven increases reusability and takes care of most of build related tasks. It helps in bypassing my steps like adding jars to the project library, building reports, executing Junits test cases, creating Jar, War Ear files for the project deployment and many more. A very significant aspect of Maven is the use of repositories to manage jar files.

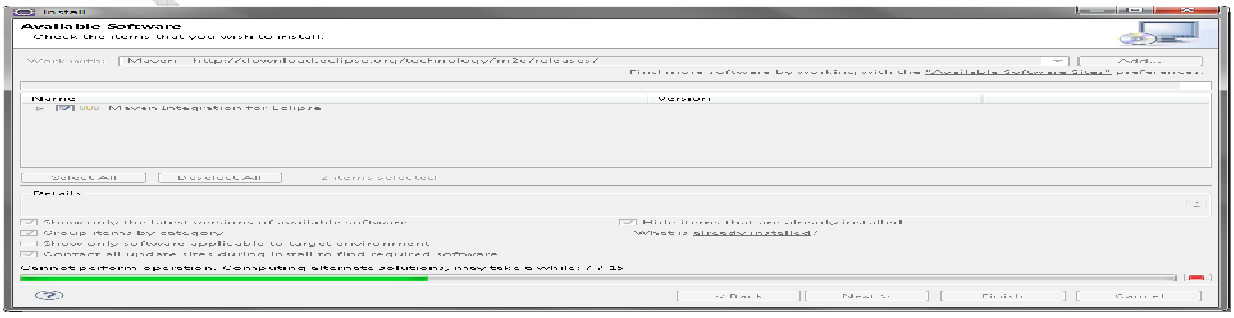
Install Maven in eclipse

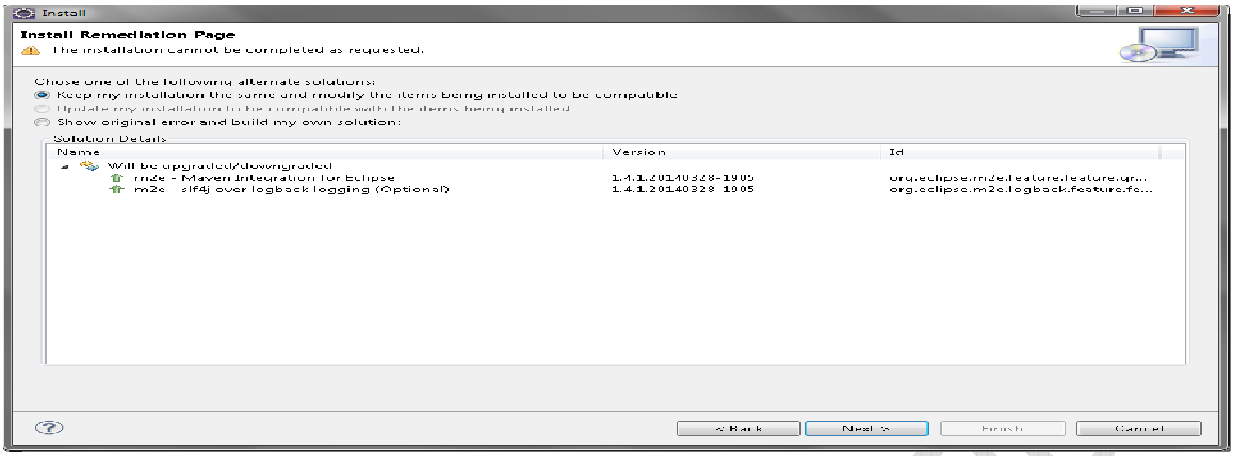


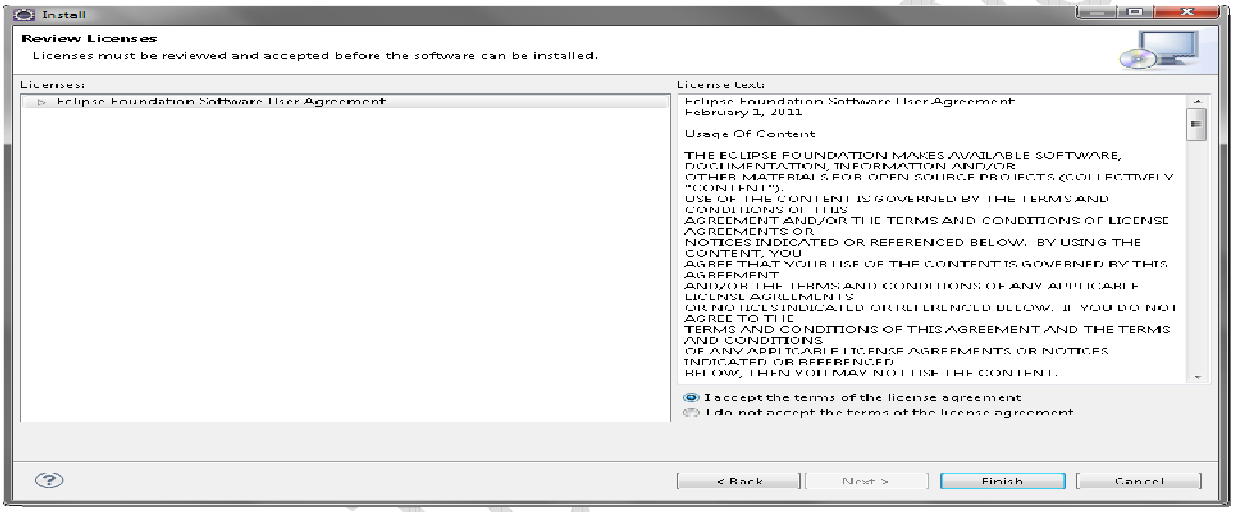









Install
Install Remediation Page
The installation cannot be completed as requested.
Choose one of the following alternate solutions:
Keep my installation the same and modify the items being installed to be compatible
Upgrade my installation to be compatible with the items being installed
Show original error and build my own solution:
Solution Details
Name
Version
Id
Will be upgraded/downgraded:
m2e - Maven Integration for Eclipse
1.4.1.20140328-1905
org.eclipse.m2e.feature.feature.gr...
m2e - slf4j over logback logging (Optional)
1.4.1.20140328-1905
org.eclipse.m2e.logback.feature.fe...
< Back
Next >
Finish
Cancel


Install
Review Licenses
Licenses must be reviewed and accepted before the software can be installed.
Licenses:
Eclipse Foundation Software User Agreement
License text:
Eclipse Foundation Software User Agreement
February 1, 2011
Usage Of Content
THE ECLIPSE FOUNDATION MAKES AVAILABLE SOFTWARE,
DOCUMENTATION, INFORMATION AND/OR
OTHER MATERIALS FOR OPEN SOURCE PROJECTS (COLLECTIVELY
"CONTENT").
USE OF THE CONTENT IS GOVERNED BY THE TERMS AND
CONDITIONS OF THIS
AGREEMENT AND/OR THE TERMS AND CONDITIONS OF LICENSE
AGREEMENTS OR
NOTICES INDICATED OR REFERENCED BELOW. BY USING THE
CONTENT, YOU
AGREE THAT YOUR USE OF THE CONTENT IS GOVERNED BY THIS
AGREEMENT
AND/OR THE TERMS AND CONDITIONS OF ANY APPLICABLE
LICENSE AGREEMENTS
OR NOTICES INDICATED OR REFERENCED BELOW. IF YOU DO NOT
AGREE TO THE
TERMS AND CONDITIONS OF THIS AGREEMENT AND THE TERMS
AND CONDITIONS
OF ANY APPLICABLE LICENSE AGREEMENTS OR NOTICES
INDICATED OR REFERENCED
BELOW, THEN YOU MAY NOT USE THE CONTENT.
I accept the terms of the license agreement
I do not accept the terms of the license agreement
< Back
Next >
Finish
Cancel