Selenium with Java

TestNG Framework

1

TestNG Framework

- TestNG is an advance framework designed in a way to leverage the benefits by both the developers and testers. For people already using JUnit, TestNG would seem no different with some advance features. With the commencement of the frameworks, JUnit gained an enormous popularity across the Java applications, Java developers and Java testers, with remarkably increasing the code quality.

Features of TestNG

- Support for annotations
- Support for parameterization
- Advance execution methodology that do not require test suites to be created
- Support for Data Driven Testing using Dataproviders
- Enables user to set execution priorities for the test methods
- Supports threat safe environment when executing multiple threads
- Readily supports integration with various tools and plug-ins like build tools (Ant, Maven etc
- Facilitates user with effective means of Report Generation using ReportNG

TestNG versus JUnit

- There are various advantages that make TestNG superior to JUnit. Some of them are:
- Advance and easy annotations
- Execution patterns can be set
- Concurrent execution of test scripts
- Test case dependencies can be set

TestNG Installation in Eclipse

Follow the below steps to TestNG Download and installation on eclipse:

•**<u>Step 1</u>:** Launch eclipse IDE -> Click on the Help option within the menu -> Select "Eclipse Marketplace.." option within the dropdown.

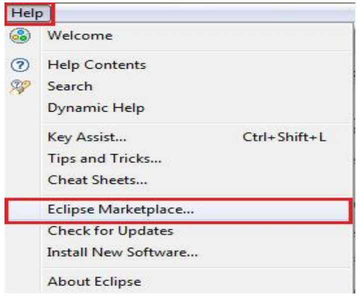
•**<u>Step 2</u>:** Enter the keyword "TestNG" in the search textbox and click on "Go" button as shown below.

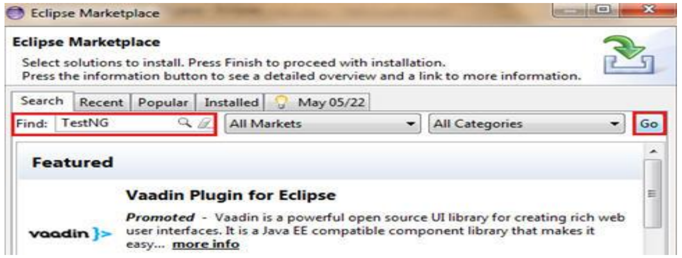
•**<u>Step 3</u>:** As soon as the user clicks on the "Go" button, the results matching to the search string would be displayed. Now user can click on the Install button to install TestNG.

TestNG installation in Eclipse

- **Step 4:** As soon as the user clicks on the Install button, the user is prompted with a window to confirm the installation. Click on “Confirm” button.
- **Step 5:** In the next step, the application would prompt you to accept the license and then click on the “Finish” button.
- **Step 6:** The installation is initiated now and the progress can be seen as following:

TestNG installation



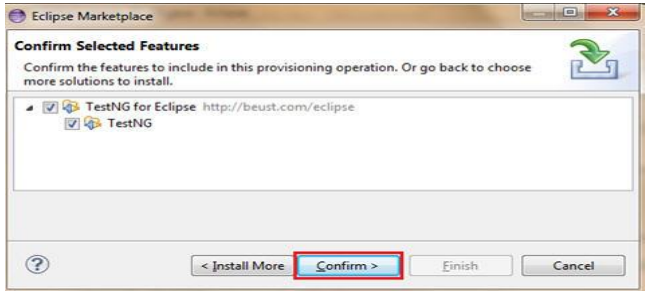


Eclipse Marketplace

Select solutions to install. Press Finish to proceed with installation.
Press the information button to see a detailed overview and a link to more information.

Vaadin Plugin for Eclipse

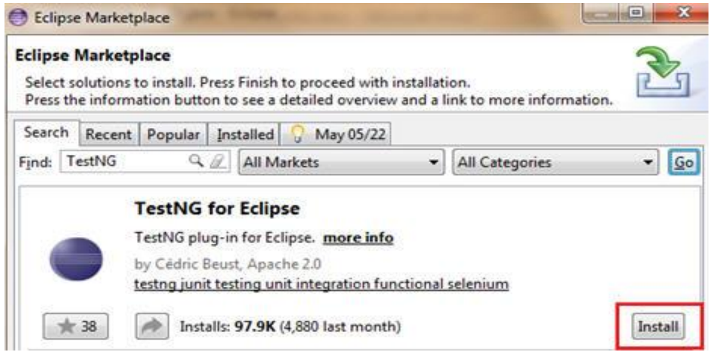
Promoted - Vaadin is a powerful open source UI library for creating rich web user interfaces. It is a Java EE compatible component library that makes it easy... more info



Confirm Selected Features

Confirm the features to include in this provisioning operation. Or go back to choose more solutions to install.

- [✓] TestNG for Eclipse http://beust.com/eclipse
  - [✓] TestNG



Eclipse Marketplace

Select solutions to install. Press Finish to proceed with installation.
Press the information button to see a detailed overview and a link to more information.

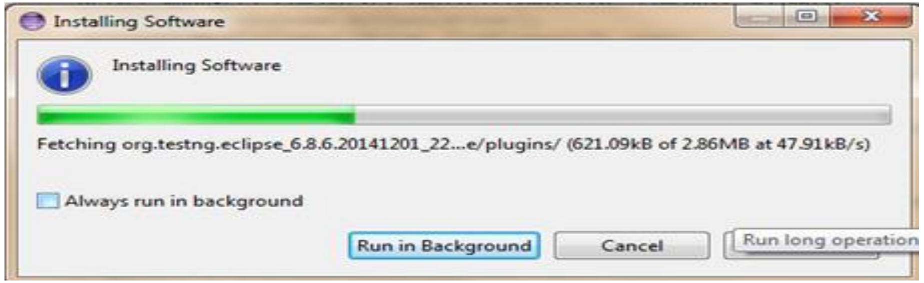
TestNG for Eclipse

TestNG plug-in for Eclipse. more info

by Cédric Beust, Apache 2.0

testng junit testing unit integration functional selenium

Installs: 97.9K (4,880 last month)



Installing Software

Fetching org.testng.eclipse_6.8.6.20141201_22...e/plugins/ (621.09kB of 2.86MB at 47.91kB/s)

TestNG Annotation

2

Test NG Annotation Description

- @BeforeSuite

  The annotated method will be run only once before all tests in this suite have run.
- @AfterSuite

The annotated method will be run only once after all tests in this suite have run.

- @BeforeClass

  The annotated method will be run only once before the first test method n the current class is invoked.
- @AfterClass

  The annotated method will be run only once after all the test methods in the current class have run.

TestNG Annotation Description

- @BeforeTest

  The annotated method will be run before any test method belonging to the classes inside the <test> tag is run.
- @AfterTest

  The annotated method will be run after all the test methods belonging to the classes inside the <test> tag have run.
- @BeforeGroups

  The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.
- @AfterGroups

  The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.

Test NG Annotation Description

- @BeforeMethod

  The annotated method will be run before each test method.
- @AfterMethod

  The annotated method will be run after each test method.
- @DataProvider

  Marks a method as supplying data for a test method. The annotated method must return an Object[ ][ ], where each Object[ ] can be assigned the parameter list of the test method. The @Test method that wants to receive data from this DataProvider needs to use a dataProvider name equals to the name of this annotation.

Test NG Annotation Description

- @Factory

  Marks a method as a factory that returns objects that will be used by TestNG as Test classes. The method must return Object[ ].
- @Listeners

  Defines listeners on a test class.
- @Parameters

  Describes how to pass parameters to a @Test method.
- @Test

  Marks a class or a method as a part of the test.

Example

let’s create a sample class whose methods will be unit-tested using testNG. Below is the sample class we will use in this example.

```
public class Calculator {
   public double add(double a, double b){
        return a+b;
    }

    public double subtract(double a, double b){
        return a-b;
    }

    public double multiply(double a, double b){
        return a*b;
    }
  }
```

Example

- Nothing special about this Calculator class, which provides methods for common arithmetic operations. We will unit test each of these methods.

```
package com.myapp.testng;
public class TestCalculator {
     Calculator calculator;
    @BeforeClass
    public void beforeClass(){
                  System.out.println("@BeforeClass: I run only once, before first test
start.");
        calculator = new Calculator();
    }
```

Example

```
@AfterClass
    public void afterClass(){
            System.out.println("@AfterClass: I run only once, after all tests have been done.\n");
        calculator = null;
    }
    @BeforeMethod
    public void beforeEachTestMethod(Method method){
System.out.println("\n@BeforeMethod: I run before each test method. Test to be executed is : "+method.getName());
}
```

Example

```
@AfterMethod
public void afterEachTestMethod(Method method){//Parameter are optional
                System.out.println("@AfterMethod: I run after each test method. Test just executed is : "+method.getName()+"\n");
    }
@Test
    public void testAdd(){
        System.out.println("@Test add");
        Assert.assertEquals(calculator.add(2, 3), 5.0);
    }
```

Example

```
 @Test
    public void testSubtract(){
        System.out.println("@Test subtract");
        Assert.assertTrue(calculator.subtract(5, 3) > 1, "Subtract test failed");
    }
@Test
    public void testMultiply(){
        System.out.println("@Test multiply");
        Assert.assertEquals(calculator.multiply(5, 3) , 15.0);
    }

}
```

Suit Test

- In TestNG, we cannot define a suite in testing source code, but it is represented by one XML file, as suite is the feature of execution. It also allows flexible configuration of the *tests* to be run. A suite can contain one or more tests and is defined by the <suite> tag.
- <suite> is the root tag of your testng.xml. It describes a test suite, which in turn is made of several <test> sections.
- Example:
- 1.create two classes with name Test1 ,Test2 whose methods will be unit-tested using testing.
- 2.Configure the suit test in an Testng.xml file.

```
<?xml version = "1.0" encoding = "UTF-8"?>
 <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
```

Suit Test

```
<suite name = "Suite1">
 <test name = "exampletest1">
 <classes>
 <class name = "Test1" />
</classes>
</test>
 <test name = "exampletest2">
 <classes> <class name = "Test2" />
 </classes>
</test>
</suite>
```

Test Ignore

- If a test method is annotated with *@Test(enabled = false)*, then the test case that is not ready to test is bypassed.
- Example:

```
Class Demo
{
String name=“admin”;
@Test(enabled = false)
 public void testLogin()
{
System.out.println("Inside Login");
Assert.assertEquals(name,”admin”);
 }
}
```

Group Test

- Group test is a new innovative feature in TestNG, which doesn’t exist in JUnit framework. It permits you to dispatch methods into proper portions and perform sophisticated groupings of test methods.
- Not only can you declare those methods that belong to groups, but you can also specify groups that contain other groups. Then, TestNG can be invoked and asked to include a certain set of groups (or regular expressions), while excluding another set.

Dependency Test

- Sometimes, you may need to invoke methods in a test case in a particular order, or you may want to share some data and state between methods. This kind of dependency is supported by TestNG, as it supports the declaration of explicit dependencies between test methods.
- TestNG allows you to specify dependencies either with –
- Using attribute *dependsOnMethods* in @Test annotations, OR.
- Using attribute *dependsOnGroups* in @Test annotations.
- Example:

```
public class Demo2 {
        @Test
         public void test_login() {
                System.out.println("login tested");
         }
```

Dependency Test

```
                    @Test(dependsOnMethods={"test_login"})
                    public void test_compose(){
                            System.out.println("Compose tested");
                    }
}
```

Parameterized Tests

- Another interesting feature available in TestNG is parametric testing.
- Parameterized tests allow developers to run the same test over and over again using different values.
- TestNG lets you pass parameters directly to your test methods in two different ways –
  - With testng.xml
  - With Data Providers

Parameterized Test

- Example:

```
public class ParameterizedTest1
{
@Test @Parameters("myName")
 public void parameterTest(String myName)
 {
System.out.println("Parameterized value is : " + myName);
 }
}
```

Parameterized Test

Testng.xml

```
<?xml version = "1.0" encoding = "UTF-8"?>
 <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
 <suite name = "Suite1">
 <test name = "test1">
<parameter name = "myName" value="manisha"/>
 <classes>
 <class name = "ParameterizedTest1" />
</classes>
</test>
 </suite>
```

Passing Parameters with Dataproviders

- 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.), parameters can be passed using Dataproviders.
- A Data Provider is a method annotated with @DataProvider. This annotation has only one string attribute: its name. If the name is not supplied, the data provider's name automatically defaults to the method's name. A data provider returns an array of objects.
- Example:
- 1.Create a class Calculator.java
- public class Calculator {
-                                 public int add(int a,int b){
-                                 return a+b;
-         }

Passing Parameters with Dataproviders

- 2. Create Test Case Class

```
public class TestCase {
            Calculator calc = null;
  @BeforeClass
  public void create_calcinstance(){
             calc= new Calculator();
  }
            @Test(dataProvider = "dpAdd")
  public void test_Add(int a, int b, int Expected_result) {
            Assert.assertEquals(calc.add(a,b),Expected_result,"Actual result not
equal to Expected result");
}
```

Passing Parameters with Dataproviders

```
▶ @DataProvider (name= "dpAdd")
        public Object [][] getData(){
                Object[][] table = new Object[][]{
                                {10,30,40},
                                {50,10,80},
                                {20,20,40},
                                {40,50,120},
                                {34,20,54},
                                {30,40,234} };
                return table;
                }
```

Passing Parameters with Dataproviders

```
@AfterClass
        public void close_calcinstance(){
                calc=null;
        }
}
```

3

Selenium

What is Selenium

- Selenium is a free (open-source)
- used for automating web-based applications UI.
- automation across different browsers, platforms.
- use multiple programming languages like Java, C#, Python etc to create Selenium Test Scripts.
- Using Selenium, we can automate the functional tests.
- easily integrate them with Maven, Jenkins, and other build automation and continuous integration tools.
- Testing done using the Selenium testing tool is usually referred to as Selenium Testing.

Selenium History

- Selenium was originally developed by Jason Huggins in 2004 as an internal tool at ThoughtWorks
- Paul Hammant joined the team and steered the development of the second mode of operation that would later become 'Selenium Remote Control' (RC).
- In 2007, Simon Stewart at ThoughtWorks developed a superior browser automation tool called WebDriver.
- In 2009, after a meeting between the developers at the Google Test Automation Conference, it was decided to merge the two projects, and call the new project Selenium WebDriver, or Selenium 2.0.
- In 2008, Philippe Hanrigou (then at ThoughtWorks) made 'Selenium Grid', which provides a hub allowing the running of multiple Selenium tests concurrently on any number of local or remote systems, thus minimizing test execution time.

Selenium Feature

Selenium Feature

- Selenium is a free (open source) automated testing suite for web applications across different browsers(IE 6/7, Firefox, Opera, Safari 2.0+) and platforms (Windows, Linux, and Macintosh).

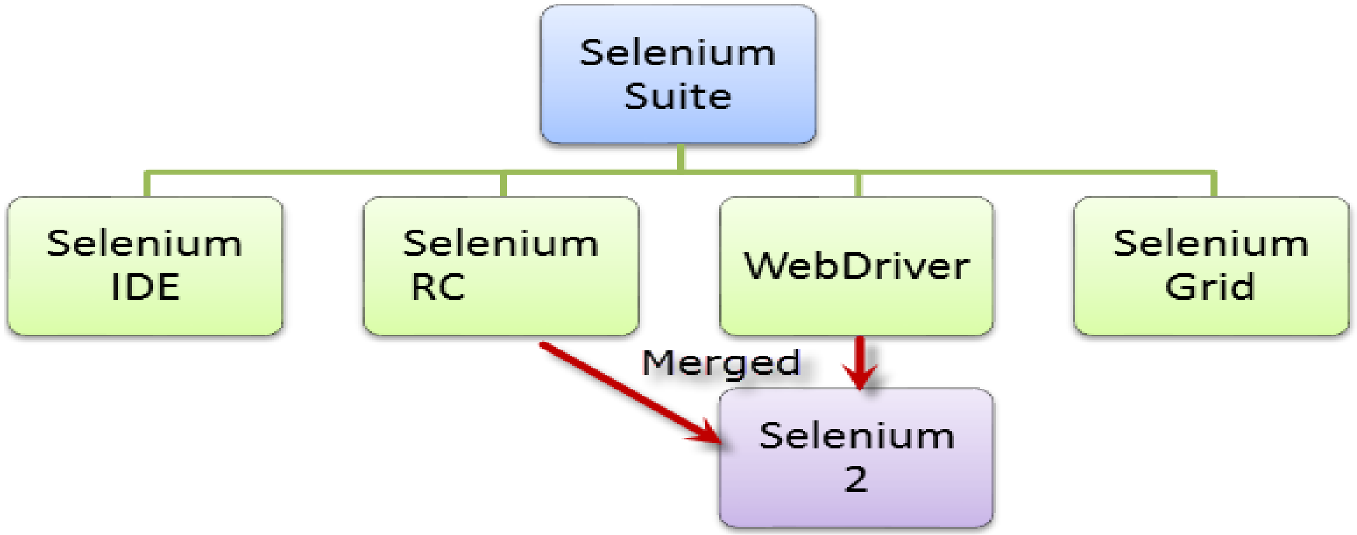
- Selenium allows scripting in several languages like Java, C#, PHP and Python.

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

5

Selenium Component

Selenium Component



Selenium Component

- Selenium IDE
  - A Firefox extension that can automate the browser through a record-and-playback feature.
  - To further increase the speed in creating test cases
- WebDriver
  - when browsers and web applications were becoming more powerful and more restrictive with JavaScript programs like Selenium Core.
  - It was the first cross-platform testing framework that could control the browser from the OS level.

Selenium Component

- Selenium Remote Control (Selenium RC)
  - Selenium RC is a server, written in java, that accepts commands for the browser via HTTP.
  - During the Early Stages testers using Selenium Core had to install the whole application under test and the web server on their own local computers
  - Later ThoughtWork's, decided to create a server that will act as an HTTP proxy to "trick" the browser into believing that Selenium Core and the web application being tested come from the same domain.

Selenium Component

- Selenium Grid
  - To address the need of minimizing test execution times as much as possible.
  - Initially called the system “Hosted QA.”
  - It was capable of capturing browser screenshots during significant stages, and also of sending out Selenium commands to different machines simultaneously

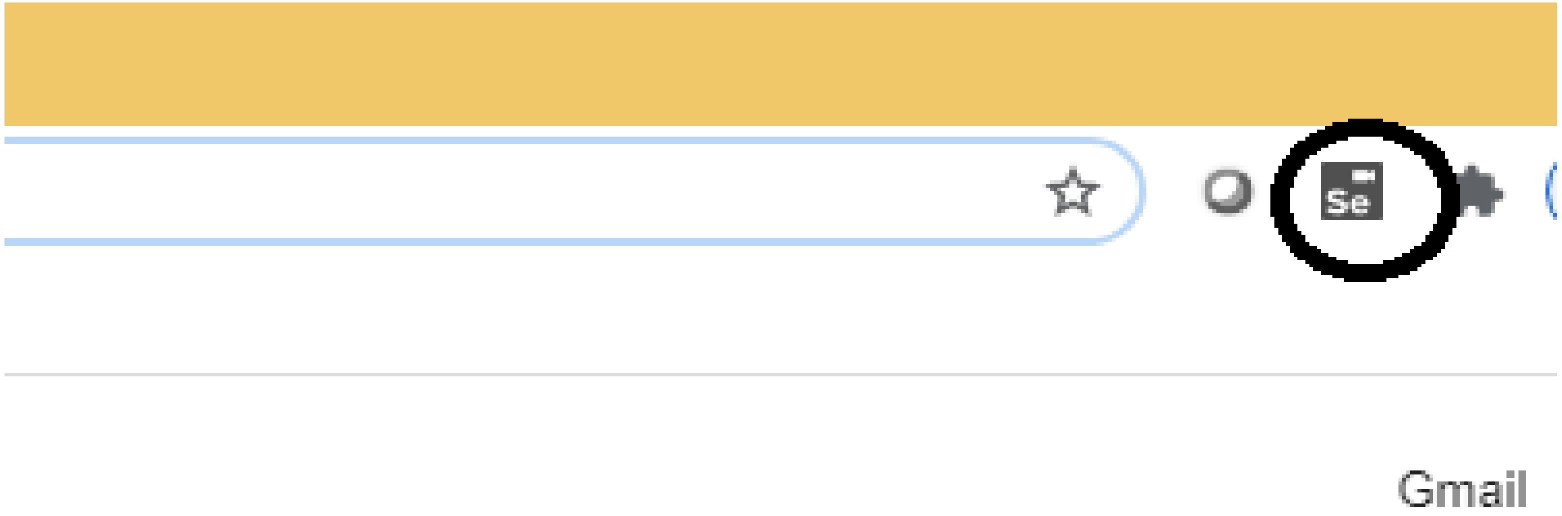
Selenium IDE

6

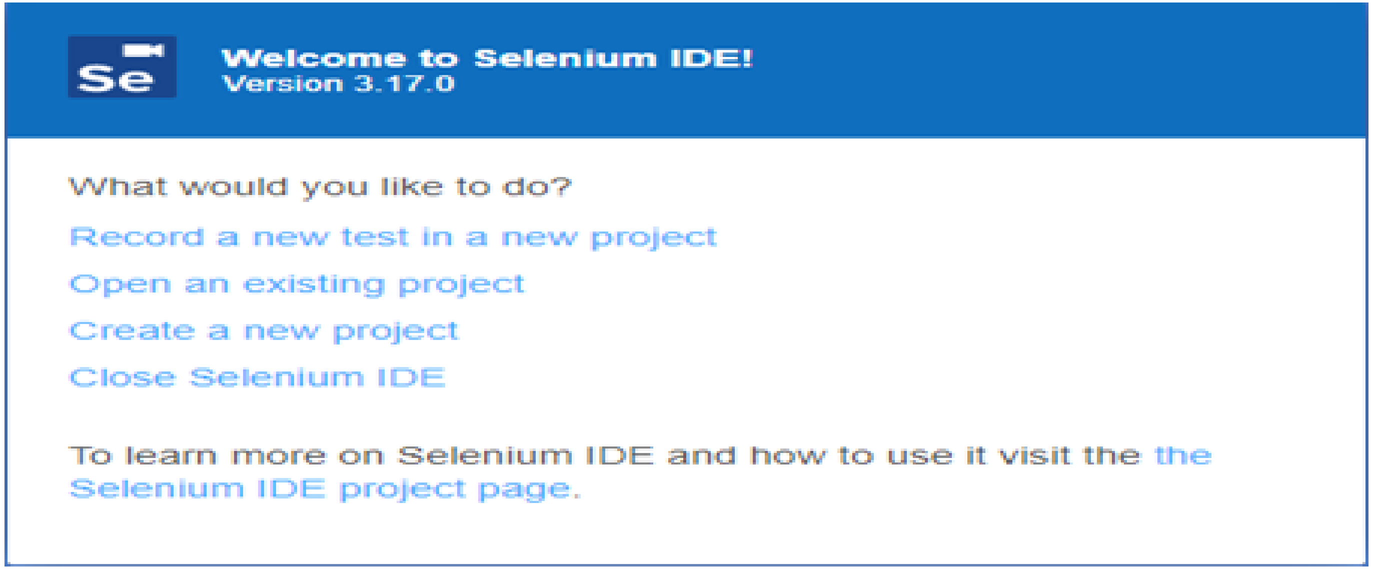
Birth Of Selenium IDE

- Shinya Kasatani of Japan created Selenium IDE, a Firefox extension that can automate the browser through a record-and-playback feature.
- He came up with this idea to further increase the speed in creating test cases.
- He donated Selenium IDE to the Selenium Project in 2006.

Selenium IDE in Chrome



Selenium IDE in Chrome


Welcome to Selenium IDE!
Version 3.17.0

What would you like to do?

Record a new test in a new project

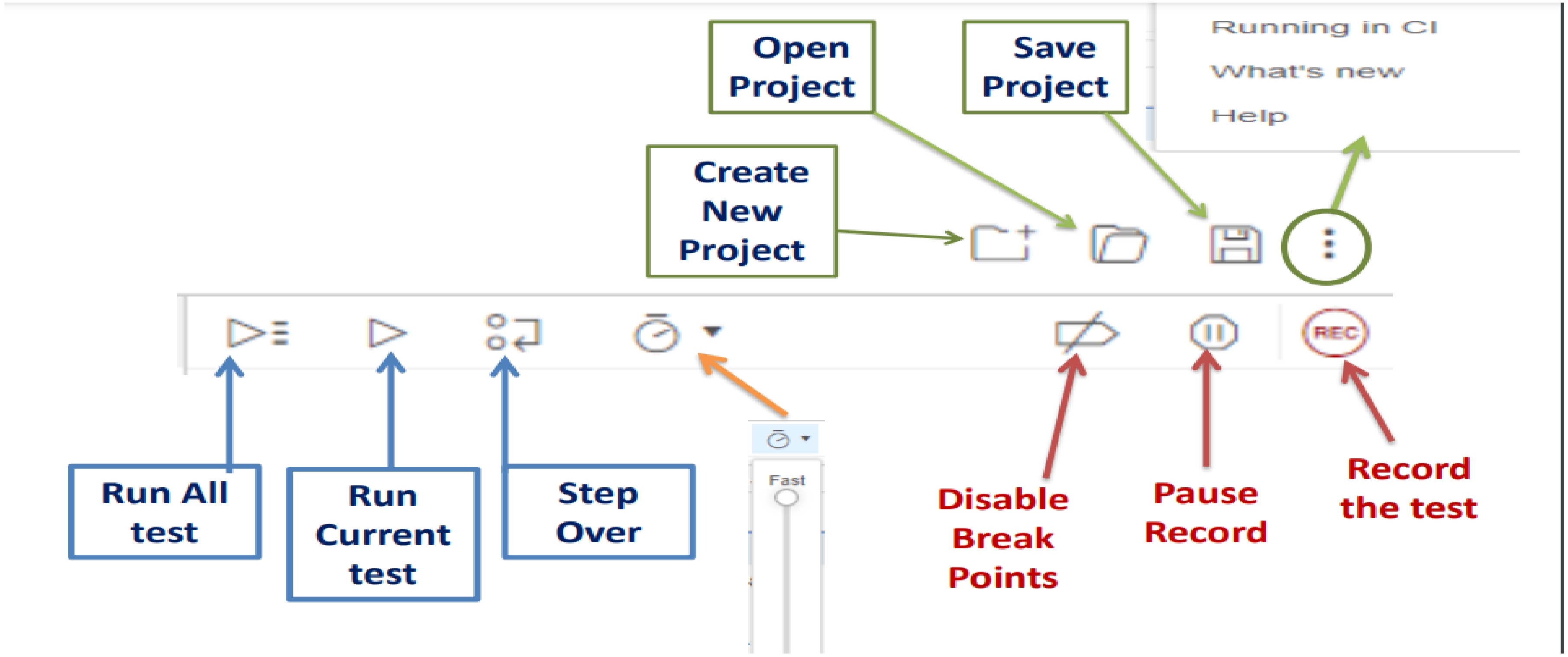
Open an existing project

Create a new project

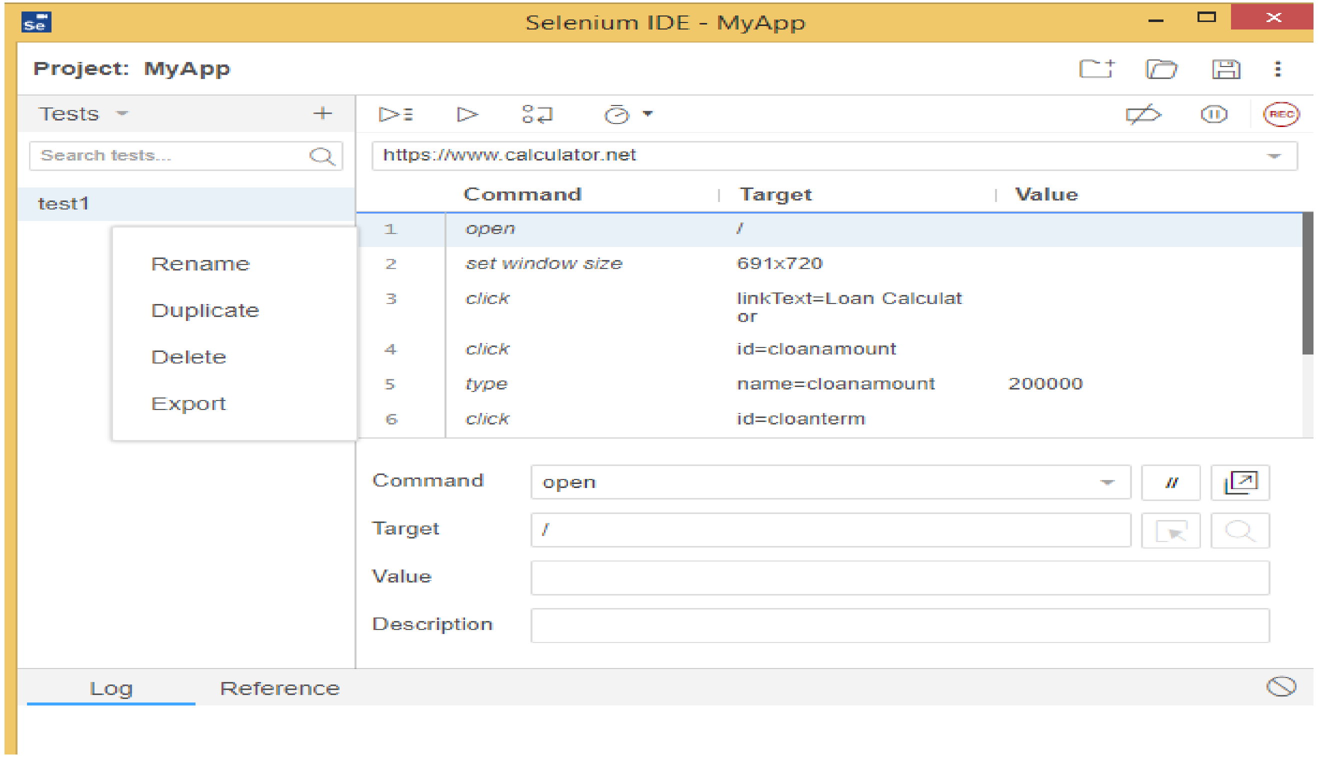
Close Selenium IDE

To learn more on Selenium IDE and how to use it visit the the Selenium IDE project page.

Selenium IDE in Chrome



Selenium IDE in chrome



https://www.calculator.net

	Command	Target	Value
1	*open*	/	
2	*set window size*	691x720	
3	*click*	linkText=Loan Calculator	
4	*click*	id=cloanamount	
5	*type*	name=cloanamount	200000
6	*click*	id=cloanterm	

Command: open
Target: /
Value:
Description:

Creating Selenium IDE Tests

- Steps are involved in creating Selenium tests using IDE:
  - Recording a test
  - adding commands in
  - Saving the recorded test
  - Saving the test suite
  - Executing the recorded test
  - Export the test script

What is Selenese ?

- Selenese is the language used to write Selenium Commands.
- These Selenese commands are then used to test web-applications.
- Based on the HTML tags of the UI elements, one can check their existence.
- Commands help Selenium understand what actions or operations to perform.

Classification of Selenium Commands

- Selenium commands are mainly categorized into three types: Actions – Help manipulate or change the state of applications (e.g. click on some link or select an option from a page).
- Accessors – Enable verification and storage of the application state (e.g. consider command “storeTextPresent” – if the text is found on the page, then it stores True else stores false).
- Assertions – Help compare expected and actual results. They act like checkpoints and if both the values are equal, only then the test case passes or else it fails. Thus, Assertions help verify whether the state of the application after executing the test case conforms to the desired state (e.g. VerifyText, waitForPageToLoad).

Classification of Selenium Commands

- Assertions have three modes:
  - Assert
  - Verify
  - WaitFor

Selenium - IDE Verification Points

- The test cases that we develop also need to check the properties of a web page.
- It requires assert and verify commands.
- There are two ways to insert verification points into the script.
- To insert a verification point in recording mode, "Right click" on the element and choose "Insert New Command". The mostly used verification commands
  - verifyElementPresent
  - assertElementPresent
  - verifyElementNotPresent
  - assertElementNotPresent
  - verifyText
  - assertText

Selenium - IDE Verification Points

- verifyChecked
- assertChecked
- verifyAlert
- assertAlert
- verifyTitle
- assertTitle
- verifyAlert
- assertAlert
- verifyTitle
- assertTitle

Selenium IDE - Synchronization Points

- During script execution, the application might respond based on server load, hence it is required for the application and script to be in sync. Given below are few a commands that we can use to ensure that the script and application are in sync.
  - waitForAlertNotPresent
  - waitForAlertPresent
  - waitForElementPresent
  - waitForElementNotPresent
  - waitForTextPresent
  - waitForTextNotPresent
  - waitForPageToLoad
  - waitForFrameToLoad

Selenium WebDriver

Selenium WebDriver

- WebDriver is a web automation framework that allows you to execute your tests against different browsers, not just Firefox (unlike Selenium IDE).
- WebDriver also enables you to use a programming language in creating your test scripts
- Following programming languages are supported by WebDriver
  - Java
  - .NetP
  - Perl
  - Ruby
  - Python
  - PHP

WebDriver API Commands and Operations

- Creating browser instance for firefox.
  - WebDriver driver= new FirefoxDriver();
- If you are using firefox 47 and so on then use Geckodriver
- Creating browser instance using Geckodriver
  - System.setProperty("webdriver.firefox.marionette","C:\\geckodriver.exe");
    WebDriver driver = new FirefoxDriver();

WebDriver Commands

- Browser Commands
- Navigation Commands
- WebElement Commands
- Switch Commands
- Wait Commands

Browser Commands

- Get Command

  get(String arg0) : void – This method Load a new web page in the current browser window. Accepts String as a parameter and returns nothing.

  Command – driver.get(appUrl);

- Get Title Command

getTitle() : String – This method fetches the Title of the current page. Accepts nothing as a parameter and returns a String value.

Command – driver.getTitle();

- Get Current URL Command

  getCurrentUrl() : String – This method fetches the string representing the Current URL which is opened in the browser. Accepts nothing as a parameter and returns a String value.

  Command – driver.getCurrentTitle();

- Get Page Source Command

getPageSource() : String – This method returns the Source Code of the page. Accepts nothing as a parameter and returns a String value.

  Command – driver.getPageSource();

Browser Commands

- Closing Browser
  - close() : void – This method Close only the current window the WebDriver is currently controlling. Accepts nothing as a parameter and returns nothing.
  - driver.close();
- Closing Browser and other all other windows associated with the driver
  - quit() : void – This method Closes all windows opened by the WebDriver. Accepts nothing as a parameter and returns nothing.
  - driver.quit();

Browser Commands

- Moving Between Windows and Frames

  Some web applications have many frames or multiple windows. WebDriver supports moving between named windows using the “switchTo” method.
- Syntax:

```
        driver.switchTo().window( "windowName" );
        driver.switchTo().frame( “frameName" );
```

Browser Navigation Commands

- Browser Navigation
- Syntax :

```
        driver.navigate().to( "http://www.example.com" );
```

- “navigate” interface also exposes the ability to move backwards and forwards in your browser’s and refresh the current page
- Syntax :

```
        driver.navigate().to("www.amazon.com");
        driver.navigate().forward();
        driver.navigate().back();
        driver.navigate().refresh();
```

WebElement Commands

- WebElement represents an HTML element. HTML documents are made up by HTML elements. HTML elements are written with a start tag, with an end tag, with the content in between
  - <p> My first HTML paragraph. </p>
- to get the WebElement object write the below statement:
  - WebElement element = driver.findElement(By.id(“UserName“));

Clear Command

- clear( ) : void – If this element is a text entry element, this will clear the value. This method accepts nothing as a parameter and returns nothing.
- Command – element.clear();

- WebElement element = driver.findElement(By.id("UserName"));
- element.clear();

▶ IsSelected Command

- isSelected( ) : boolean – Determine whether or not this element is selected or not. This accepts nothing as a parameter but returns boolean value(true/false).
- Command – element.isSelected();

WebElement Commands

- Submit Command
  - submit( ) : void– This method works well/better than the click() if the current element is a form, or an element within a form. This accepts nothing as a parameter and returns nothing.
  - Command – element.submit();
- SendKeys Command
  - sendKeys(CharSequence… keysToSend) : void – This simulate typing into an element, which may set its value. This method accepts CharSequence as a parameter and returns nothing.
  - Command – element.sendKeys(“text”);

WebElement Commands

- Click Command
  - click( ) : void – This simulates the clicking of any element. Accepts nothing as a parameter and returns nothing.
  - Command – element.click();
- IsEnabled Command
  - isEnabled( ) : boolean – This determines if the element currently is Enabled or not? This accepts nothing as a parameter but returns boolean value(true/false).
  - Command – element.isEnabled();

What Are Browser Elements?

- Elements are the different components that are present on web pages. The most common elements we notice while browsing are:
  - Text boxes
  - Buttons
  - Images
  - Hyperlinks
  - Radio buttons/ Checkboxes
  - Text area/ Error messages
  - Drop down box/ List box/ Combo box
  - Web Table/ HTML Table
  - Frame

findElement(), findElements()

- There are various techniques by which the WebDriver identifies the form elements based on the different properties of the Web elements like
  - By ID
  - By Name
  - By Link Text
  - By CSS
  - By Xpath
- Web Driver provides the following two methods to find the elements.
- findElement() – finds a single web element and returns as a WebElement object.
- findElements() – returns a list of WebElement objects matching the locator criteria

Questions

Thank you