|  |  |  |
| --- | --- | --- |
| <https://www.softwaretestingclass.com/using-contains-sibling-ancestor-to-find-element-in-selenium/>Sr. No | Mandatory | Optional |
| 1 | Accessing Forms in Webdriver | How to use AutoIT with Selenium |
| 2 | How to Select Option from DropDown using Selenium Webdriver | Execute JavaScript based code using Selenium Webdriver |
| 3 | Accessing Links & Tables using Selenium Webdriver | How to use intelliJ & Selenium Webdriver |
| 4 | Keyboard & Mouse Event using Action Class in Selenium Webdriver | Handling Ajax call in Selenium Webdriver |
| 5 | How to Upload & Download a File using Selenium Webdriver | Flash Testing with Selenium |
| 6 | Xpath. ,Locators  in Selenium: Complete Guide | How to Find Broken links using Selenium Webdriver |
| 7 | How to Install TestNG in Eclipse for Selenium WebDriver | Selenium Core Extensions |
| 9 | How to Execute TestNG Project & Failed Test Cases by Command Prompt | Using Apache Ant with Selenium |
| 10 | Handling Date Time Picker using Selenium | Using Selenium with Github |
| 11 | Alert & Popup handling in Selenium | Using SoapUI with Selenium |
| 12 | Handling Dynamic Web Tables Using Selenium WebDriver | XSLT Report in Selenium |
| 13 | Using Contains, Sibling, Ancestor to Find Element in Selenium | Firefox Profile - Selenium WebDriver |
| 14 | Implicit & Explicit Waits in Selenium | Breakpoints and Startpoints in Selenium |
| 15 | All About Excel in Selenium: POI & JXL | Drag and Drop action in Selenium |
| 16 | Page Object Model (POM) & Page Factory in Selenium: Ultimate Guide | File Upload using Sikuli in Selenium Webdriver |
| 17 | Introduction to Selenium Grid |  |
| 18 | Maven & Jenkins with Selenium: Complete Tutorial |  |
| 19 | Creating Keyword & Hybrid Frameworks with Selenium |  |
| 20 | Database Testing using Selenium: Step by Step Guide |  |
| 21 | Handling Iframes in Selenium |  |
| 22 | Cross Browser Testing using Selenium |  |
| 23 | PDF , Emails and Screenshot of Test Reports in Selenium |  |
| 24 | How to Take Screenshot in Selenium WebDriver |  |
| 25 | Sessions, Parallel run and Dependency in Selenium |  |
| 26 | Tutorial on Log4j and LogExpert with Selenium |  |
| 27 | Selenium with HTMLUnit Driver & PhantomJS |  |
| 28 | Using Robot API with Selenium |  |
| 29 | Desired Capabilities in Selenium |  |
| 30 | SSL Certificate Error Handling in Selenium |  |
| 31 | Listeners and their use in Selenium WebDriver |  |
| 32 | TestNG: Execute multiple test suites |  |
| 33 | Introduction to TestNG Groups |  |
| 34 | Handling Cookies in Selenium WebDriver |  |
| 36 | Using Cucumber with Selenium |  |
| 37 | Creating Object Repository in Selenium WebDriver |  |
| 38 | Scroll UP or Down a page in Selenium Webdriver |  |
| 39 | Find Element and Find Elements in Selenium |  |

Optional:

|  |
| --- |
| How to use AutoIT with Selenium |
| Execute JavaScript based code using Selenium Webdriver |
| How to use intelliJ & Selenium Webdriver |
| Handling Ajax call in Selenium Webdriver |
| Flash Testing with Selenium |
| How to Find Broken links using Selenium Webdriver |
| Selenium Core Extensions |
| Using Apache Ant with Selenium |
| Using Selenium with Github |
| Using SoapUI with Selenium |
| XSLT Report in Selenium |
| Firefox Profile - Selenium WebDriver |
| Breakpoints and Startpoints in Selenium |
| Drag and Drop action in Selenium |
| File Upload using Sikuli in Selenium Webdriver |

What is selenium?

Its open source tool

for web applications

Set of software to automate web browser.

Regression testing & functional testing

Supports provided by Selenium :

Operating environments:

Mac, windows, linux etc...

Browsers:

Mozilla,IE, Google chrome,safari, opera etc...

Programming language support:

Java,c#, perl,python,Ruby,PHP etc..

Introduction to TestNG

•             Install and set up TestNG environment

•             TestNG annotations

•             Groups & DependOn

•             Group Test in TestNG

•             Using Testng.xml in TestNG

•             TestNG Data Providers.

Selenium Web Driver

•             Selenium WebDriver Introduction

•             Configuration

•             Creation of test scripts using web driver commands.

•             Cross Browser Testing –Firefox, IE, Chrome

•             Web driver commands to access basic GUI objects

•             Selenese Commands

•             Using Locators in Selenium

•             Synchronization in Selenium Web Driver

•             Handling Web Tables, Frames

•             Handle Alerts and Multiple Windows using WebDriver

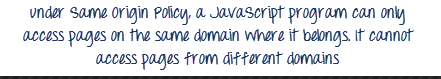
Selenium is not just a single tool but a suite of software's, each catering to different testing needs of an organization. **It has four components.**

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

History:

Primarily, Selenium was **created by Jason Huggins in 2004**. An engineer at ThoughtWorks, he was working on a web application that required frequent testing. Having realized that the repetitious [Manual Testing](https://www.guru99.com/manual-testing.html) of their application was becoming more and more inefficient, he created a[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)program that would automatically control the browser's actions. He named this program as the "**JavaScriptTestRunner**.".

Drawback:



This is the reason why prior to Selenium RC, testers needed to install local copies of both Selenium Core (a JavaScript program) and the web server containing the web application being tested so they would belong to the same domain

Unfortunately; testers using Selenium Core had to install the whole application under test and the web server on their own local computers because of the restrictions imposed by the **same origin policy.**So another ThoughtWork's engineer, **Paul Hammant**, 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. This system became known as the **Selenium Remote Control** or **Selenium 1**.

RC:

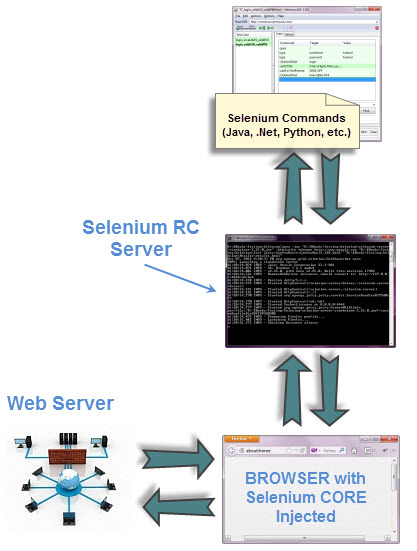
Selenium-RC leverages the full power of programming languages, creating tests that can do things like read and write external files, make queries to a database, send emails with test reports, and practically anything else a user can do with a normal application.

Selenium-RC is composed of two parts:

* The Selenium Server which launches and kills browsers, and acts as an *HTTP proxy* for browser requests.
* Client libraries for various programming languages, each of which instructs the Selenium Server in how to test the AUT by passing it your test script’s Selenium commands.

the client libraries communicate with the Server passing each Selenium command for execution. Then the server passes the Selenium command to the browser using Selenium-Core JavaScript commands. The browser, using its JavaScript interpreter, executes the Selenium command, which effectively, runs the check you specified in your Selenese test script.

The RC server bundles Selenium Core, and then automatically injects it into the browser. This occurs when your test program causes the browser to open (using a client library API function). Selenium-Core is a JavaScript program, actually a set of JavaScript functions, which interprets and executes Selenese commands using the browser’s built-in JavaScript interpreter.



Selenium web driver:

provides programming interface to create & execute TCs.

* IDE HAS IDE(RECORD & REPLAY) But no programming interface
* WD Has programming interface but does not have IDE.
* It supports multiple OE like mac, linux,windows.
* It supports diffrent browsers, chrome, mozilla, IE, safari, Opera etc.
* It also supports pogramming environments like, java, c# , perl, python, php etc..

limitations :

* No IDE
* Does not support test suit execution.
* Details reports will not be generated using WD.
* NO centralised maintenance of elements.
* DATA driven testing not possible.

Tools used in selenium:

* JUnit
* TestNG Framework
* Eclipse
* JDK 1.6 <
* Selenium Web driver jar files
* IDE Plugin
* Firebug & firepath plugin's for inspecting elements in firefox.
* IE OR Chrome browser f12 used to inspect the elements.
* browser drivers will be needed to execute scripts on multiple borwsers.

[www.seleniumhq.org](http://www.seleniumhq.org)

Selenium UFT

1. Open source Paid tool (licensed)

2. Supports various programming supports only vb for scripting

languages

3. suitable for web based app supports web, windows, mobile app

4. Supports diffrent Browsers recomonded to work with IE

5. supports diffrent platforms Can work with Windows

OS

6. No Addins needed Addins needed

7. Integration not posible with Can integrate with ALM/QC tools

other tools

8. No cetralised maintenace Centralised Object repository available

for elements

9. Complicated setup essy to install

10.detail programming knowledge Easy to learn VB & tool

needed

11. Less hardware resources More hardware resources used

needed

12. Limited support for image provides rich support for iMAGE Based tests based testing

13. No certification Certifications available

14. No dedicated recomonded dedicated help / support by HP. help available

Selenium versions.

Selenium 1 = Selenium Remote Control. Selenium 2 = Selenium Webdriver, which combines elements of Selenium 1 and Webdriver.

The big downside of Selenium RC was that it was written in JavaScript, so depended on the browser's JavaScript engine as to how it would run. This made tests buggy.

Selenium 2 integrated Google's Webdriver project. It's an interface that runs the JavaScript natively, so you only need to write the test once and Selenium will test on different browsers.

Therefore, Selenium 1 works with just about every browser - if it has a JavaScript engine, it will probably work. However, Selenium 2 needs a driver for that browser. Although there are drivers for all the biggest browsers, for testing the maximum range of browser / OS combinations, it is better to use Selenium 1. If you are just testing the most common browsers, Selenium 2 is more accurate.

## Selenium 3 Important Features

Some of the most important features of Selenium 3 are given in the below list. Due to these new features, there have been many changes in the way you setup and use Selenium. We have taken utmost care that we incorporate all these changes in our articles & code snippets, so that you don’t face any issues while downloading and setting up the latest versions of Selenium WebDriver.

* Minimum version of Java required to run Selenium 3 is Java 8+
* You would need to use GeckoDriver if you want to run your scripts in Firefox versions greater than 47.0.1. Firefox 47.0.1 and before would not need GeckoDriver
* Apple has come up with its own SafariDriver to let you run your tests in Safari on Mac
* Guava has been updated to version 21

# Move Over, Selenium 2: Why It’s Time to Upgrade to Selenium 3

Posted March 30, 2017 by Greg Sypolt in [Selenium](https://saucelabs.com/blog/category/selenium)[Selenium Resources](https://saucelabs.com/blog/category/selenium-resources)

Move over, [Selenium](http://www.seleniumhq.org/) 2. It's time to upgrade to Selenium 3.

I have always been an early adopter. I can't help it. So it’s hard to believe it has been about six months since Selenium 3 released its first beta version, and that I’ve only recently been able to take it for a test drive.

The upgrade from Selenium 2 to Selenium 3 was practically effortless. (I recommend that people start a proof of concept project to demonstrate that Selenium 3 works with their existing suite of automated tests.) I expect your project may require some tweaking, and then you'll be up and running again. Take some time to explore all the new features, review known issues, and what is on the roadmap for Selenium 3. At the end of the day, you will need to evaluate if the upgrade is feasible and that it will not break your current Selenium testing project.

### What’s New and What’s Changed with Selenium 3

Any major version change typically means significant additions in functionality, or interface changes. It is important to review the release notes before upgrading your test suites blindly. After hours of reading Selenium release notes, reading blogs, and viewing a webinar by Simon Stewart, the Selenium Project Lead, I have a better understanding of what has changed with the Selenium 3 release. It’s possible that I overlooked something, but here is a compiled list of changes I took note of:

* Java 8 is required.
* The original Selenium Core will be retired, which has a significant impact for teams.
* The Selenium RC APIs have been moved to a “legacy” package, a.k.a. leg-rc
* The support for browser drivers has changed.

|  |  |
| --- | --- |
| **Google** | * Will provide their own [chromedriver](https://sites.google.com/a/chromium.org/chromedriver/) binary |
| **Mozilla** | * Upgrading to Selenium 3 no longer supports the default Mozilla Firefox browser—now onwards to Mozilla's [geckodriver](https://github.com/mozilla/geckodriver/releases) |
| **Apple** | * Safari browser is provided on macOS (Sierra 10.12 or later), and Apple owns the [safaridriver](https://developer.apple.com/library/prerelease/content/releasenotes/General/WhatsNewInSafari/Articles/Safari_10_0.html). * Selenium project will no longer maintain the OSS SafariDriver once Safari 10 ships. |
| **Microsoft** | * Edge browser is provided by Microsoft from their [webdriver server](https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/). * IE versions 9 or above are supported. |

* Evolving W3C webdriver specification
* Stability fixes for Selenium Grid
* Bug fixes

Since day one, the Selenium project has been responsible for providing the webdriver for each browser. (A massive effort by non-paid project contributors maintains the webdriver for each browser.)

Now, the major browser vendors ([Google Chrome](https://sites.google.com/a/chromium.org/chromedriver/), [Apple Safari](https://webkit.org/blog/6900/webdriver-support-in-safari-10/), [Microsoft Edge](https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/), and [Mozilla Firefox](https://github.com/mozilla/geckodriver/releases)) are shipping their own implementation of the webdriver. This change makes complete sense, as the browser vendors understand their product better than anyone else, and have full access to the source code. The movement benefits all Selenium users now that browser vendors are eating their own dog food! (“Eating your own dog food” means that a company is using its own products or services for internal operation. In this case, I hope they are developing Selenium end-to-end tests to validate their own webdriver implementation.)

### On the Roadmap—W3C Spec

A future Selenium release is already in motion. The Selenium team has been working hard on getting an official webdriver specification published by W3C.

W3C develops common protocols, guidelines, and open standards that ensure long-term growth for the Web will evolve in the same direction, rather than splitting in different directions. We need W3C webdriver specification because browsers are complex, and delivering a rock-solid specification that all browser vendors must follow will have a positive impact on the testing community. Browser vendors which own their drivers must prepare for future [W3C](https://www.w3.org/) published webdriver specifications.

### Why It’s Time to Upgrade

It’s critical that Selenium Core users start planning a strategy for upgrading to the latest version of Selenium, and start using the webdriver API. For the testing community using the Selenium webdriver API, it will be a seamless upgrade from Selenium 2.53.1 to Selenium 3.x.x. (The biggest change for me was downloading geckodriver for local testing.) And it has been a smooth transition for remote testing using Sauce Labs.

#### Sauce Labs Platform and Browser Compatibility with Selenium 3

|  |  |
| --- | --- |
| **Windows 10 MacOS 10.11 MacOS 10.12** | Edge 14.14393 >= Chrome 53 >= Firefox 49 >= IE v 11.103 >= Safari 9.0 |
| **MacOS 10.12** | Safari 10.0 |

The advantages of upgrading include some bug fixes, improved stability, browser vendors starting to own implementation of their own drivers, and more. If you encounter any issues with Selenium 3, I recommend checking out the reported Selenium GitHub [issues](https://github.com/SeleniumHQ/selenium/issues) for the project.

Selenium IDE[edit]

Selenium IDE is a complete integrated development environment (IDE) for Selenium tests. It is implemented as a Firefox Add-On and as a Chrome Extension. It and allows for recording, editing, and debugging of functional tests. It was previously known as Selenium Recorder. Selenium-IDE was originally created by Shinya Kasatani and donated to the Selenium project in 2006. Selenium IDE was previously little-maintained.[4] Selenium IDE began being actively maintained in 2018.[5]

Scripts may be automatically recorded and edited manually providing autocompletion support and the ability to move commands around quickly. Scripts are recorded in Selenese, a special test scripting language for Selenium. Selenese provides commands for performing actions in a browser (click a link, select an option), and for retrieving data from the resulting pages.

The 2.x version of the Selenium IDE for Firefox stopped working[6] after the Firefox 55 upgrade and has been replaced by Selenium IDE 3.x.[7] However users can run the older Selenium IDE on some older Firefox versions (pre-Firefox 55) or try other alternative solutions [8]

testNG:

next generation of JUNit

configure TestNG:

Install eclipse - > help-> eclipse marketplace-> serch for TestNG-> Install

To confirm installation:

Goto help-> install new-> already installed softwares-> testNG will be displayed

To configure TestNG library:

We have to create project-> rclick on project-> build path-> configure build path-> library-> add library-> select testNG-> open

TestNG Features

* Supports annotations.
* TestNG uses more Java and OO features.
* Supports testing integrated classes (e.g., by default, no need to create a new test class instance for every test method).
* Separates compile-time test code from run-time configuration/data info.
* Flexible runtime configuration.
* Introduces ‘test groups’. Once you have compiled your tests, you can just ask TestNG to run all the "front-end" tests, or "fast", "slow", "database" tests, etc.
* Supports Dependent test methods, parallel testing, load testing, and partial failure.
* Flexible plug-in API.
* Support for multi threaded testing.

Annotains

@Test,@BeforeTest,@AfterTest,@BeforeMethod,@AfterMethod,@BeforeClass,@AfterClass,@Dataprovider.

Priority,dependsOnMethod,dependsOnMethods

Alpha/priority/dependsOn

Examples:

Com.p1

Com.testNg

Grouping :

dependsOnGroup

<include>

Parameterization (com.syntel.excelDp)

Exp1:with static data

Exp2:with excel

Include and Exclude Test Methods in TestNG

[Home](https://www.seleniumeasy.com/) >> [TestNG Tutorials](https://www.seleniumeasy.com/testng-tutorials) >> [Include and Exclude Test Methods in TestNG](https://www.seleniumeasy.com/testng-tutorials/include-exclude-test-methods-in-testng)

Submitted by *harrydev* on Sun, 10/19/2014 - 13:10

TestNg provides an option to include or exclude Groups, Test Methods, Classes and Packages using include and exclude tags by defining in testng.xml.

First we will create an examples to use include and exclude tags for Test Methods in a class.

We will create a Class with three Test Methods. In that we will include two test methods and try to exclude one test method.

Below is the example class file with three test methods

package com.easy.entry;

import org.testng.annotations.Test;

public class AddTestCase {

@Test

public void addLocationTestCase() {

System.out.println("Im in add location test case");

}

@Test

public void addDepartmentTestCase() {

System.out.println("Im in add department test case");

}

@Test

public void addEmployeeTestCase() {

System.out.println("Im in add employee test case");

}

}

In the above class example, we have created three test methods, 'addLocationTestCase', 'addDepartmentTestCase', and  
'addEmployeeTestCase'.

In the below testng.xml file we will exclude 'addEmployeeTestCase' and try to execute the program. We need to first add the class name and in that class , we need to define the methods which needs to be included and excluded

<!DOCTYPE suite SYSTEM "[http://testng.org/testng-1.0.dtd"](http://testng.org/testng-1.0.dtd) >

<suite name="Sample Test Suite" verbose="1" >

  <test name="Method Test Cases" >

    <classes>

       <class name="com.easy.entry.AddTestCase">

        <methods>

        <include name="addLocationTestCase" />

        <include name="addDepartmentTestCase" />

        <exclude name="addEmployeeTestCase" />

      </methods>

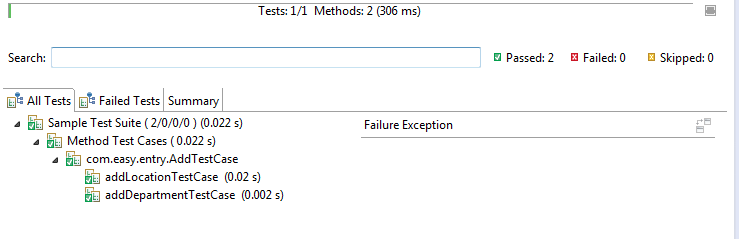
       </class>

    </classes>

  </test>

</suite>

After running the above testng.xml file, we will get the output as below. It will just run the test methods which are included in the class. And will not execute the test methods which are in excluded.



Here as in the above xml file, we can also just mention test methods which you want to exclude. If say there are 20 test methods in a class, and if you just want to exclude one or two test methods, you can just have exclude tag with test methods without again adding all the other tests methods using include tag.

Look at the below xml file to exclude only the particular test methods which are needed. In the below example, we are trying to exclude only one test method. And all the test methods in a class will get executed by default.

<!DOCTYPE suite SYSTEM "[http://testng.org/testng-1.0.dtd"](http://testng.org/testng-1.0.dtd) >

<suite name="Sample Test Suite" verbose="1" >

  <test name="Method Test Cases" >

    <classes>

       <class name="com.easy.entry.AddTestCase">

        <methods>

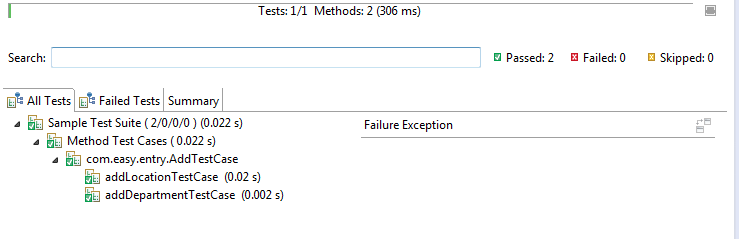
        <exclude name="addEmployeeTestCase" />

      </methods>

       </class>

    </classes>

Below is the same output which is generated.



Here again same as above, if you just want to execute only one or two test methods, we can simply mention only those test methods in the testng.xml file. Only those test methods will be executed and all the rest test methods will not be executed.

Look at the below xml file to include only particular test methods

<!DOCTYPE suite SYSTEM "[http://testng.org/testng-1.0.dtd"](http://testng.org/testng-1.0.dtd) >

<suite name="Sample Test Suite" verbose="1" >

  <test name="Method Test Cases" >

    <classes>

       <class name="com.easy.entry.AddTestCase">

        <methods>

        <include name="addDepartmentTestCase" />

      </methods>

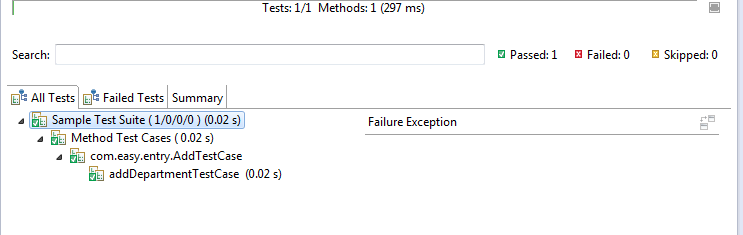
       </class>

    </classes>

  </test>

</suite>

Below is the same output which is generated. Even though there are three test methods in the class, we are trying to execute only one particular test method by specifying the test method in include tag.



Selenium WebDriver

Eclipse

JDK

Java lang binding jar

chrome or IE Drivers

Demo1: Browser,Max,close,open,get ,URLrefresh,Navigate within pages,back,forward

FirefoxDriver driver =**new** FirefoxDriver();

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

driver.get("http:\\www.google.com");

System.***out***.println("current URL: "+driver.getCurrentUrl());

driver.navigate().to("https://webapp2.syntelinc.com");

String URL = driver.getCurrentUrl();

System.***out***.println("URL Is: "+ URL);

driver.navigate().back();

System.***out***.println("current URL: "+driver.getCurrentUrl());

driver.navigate().forward();

System.***out***.println("current URL: "+driver.getCurrentUrl()); driver.close();

Web Element:

* [Introduction to WebElement, findElement(), findElements()](https://www.guru99.com/accessing-forms-in-webdriver.html#1)
* [Input Box](https://www.guru99.com/accessing-forms-in-webdriver.html#2)
* [Entering Values in Input Boxes](https://www.guru99.com/accessing-forms-in-webdriver.html#3)
* [Deleting Values in Input Boxes](https://www.guru99.com/accessing-forms-in-webdriver.html#4)
* [Buttons](https://www.guru99.com/accessing-forms-in-webdriver.html#5)
* [Submit Buttons](https://www.guru99.com/accessing-forms-in-webdriver.html#6)
* [Radio Button](https://www.guru99.com/accessing-forms-in-webdriver.html#7)
* [Check Box](https://www.guru99.com/accessing-forms-in-webdriver.html#8)
* DropDown
* Table
* Link (https://www.guru99.com/accessing-links-tables-selenium-webdriver.html)

Keyboard & Mouse Event:

Handling keyboard events and mouse hover events using Webdriver

[Home](http://www.seleniumeasy.com/) >> [Selenium Tutorials](http://www.seleniumeasy.com/selenium-tutorials) >> [Handling keyboard events and mouse hover events using Webdriver](http://www.seleniumeasy.com/selenium-tutorials/handling-keyboard-events-and-mouse-hover-events-using-webdriver)

Submitted by *harrydev* on Wed, 03/19/2014 - 11:49

In Webdriver, handling keyboard events and mouse events (including actions such as Drag and Drop or clicking multiple elements With Control key) are done using the advanced user interactions API . It contains Actions and Action classes which are needed when performing these events.  
In order to perform action events, we need to use org.openqa.selenium.interactions.Actions class.

Here is the sample code to work with Action Class

// Configure the Action

Actions action = new Actions(driver);

// To click on the element

action.moveToElement(element).click().perform();

**Note:** We need to use perform() to execute the action.

Using Action API, keyboard interactions are simple to do with webdriver. In Advanced User Interactions API, interaction with element is possible either by clicking on element or sending a Keys using sendKeys()

To use mouse actions, we need to use current location of the element and then perform the action.

The following are the regularly used mouse and keyboard events :

**Method :clickAndHold()**  
Purpose: Clicks without releasing the current mouse location

**Method : contextClick()**  
Purpose: Performs a context-click at the current mouse location.  
[How to work with context menu by taking a simple example](http://seleniumeasy.com/selenium-tutorials/right-click-context-menu-webdriver-example)

**Method: doubleClick()**  
Purpose: Performs a double click at the current mouse location

**Method: dragAndDrop(source,target)**  
Parameters: Source and Target  
Purpose: Performs click and hold at the location of the source element and moves to the location of the target element then releases the mouse.

**Method : dragAndDropBy(source,x-offset,y-offset)**  
Parameters: Source, xOffset - horizontal move, y-Offset - vertical move Offset  
Purpose: Performs click and hold at the location of the source element moves by a given off set, then releases the mouse.

**Method: keyDown(modifier\_key)**  
Parameters: Modifier\_key (keys.ALT or Keys.SHIFT or Keys.CONROL)  
Purpose: Performs a modifier key press, doesn't release the modifier key. Subsequent interactions may assume it's kept pressed

**Method: keyUp(modifier\_key)**  
Parameters: Modifier\_key (keys.ALT or Keys.SHIFT or Keys.CONROL)  
Purpose: Performs a key release.

**Method: moveByOffset(x-offset, y-offset)**  
Parameters: X-Offset , Horizontal offset, a negative value means moving the mouse to left side.  
Y-Offset, vertical offset, a negative value means moving the mouse to up.  
Purpose: Moves the mouse position from its current position by the given offset.

Check for the example here [Resizing a web element using movebyoffset](http://seleniumeasy.com/selenium-tutorials/resizing-a-web-element-using-webdriver-action-class-movebyoffset)

**Method: moveToElement(toElement)**  
Parameters: toElement - Element to Move to  
Purpose: It moves the Mouse to the middle of the element.

**Method: release()**  
Purpose: It releases the left mouse button at the current mouse location.

**Method: sendKeys(onElement, charSequence)**  
Parameters: onElement, which will receive the keyStrokes, usually text field.  
charsequence- any string value representing the sequence of keyStrokes to be sent.  
Purpose: It sends a series of keyStrokes onto the element

Here 'build()' method is used to compile all the list of actions into a single step and ready to be performed

We need to use perform() to execute the action.

Datepicker in selenium

Alerts In Selenium:

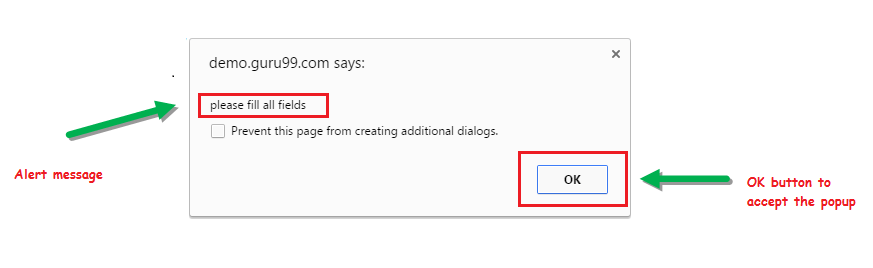
**What is Alert?**

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

Here are few alert types:

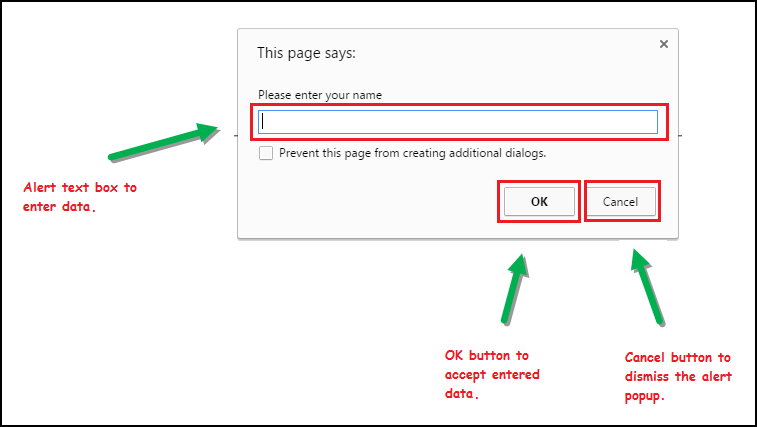
**1) Simple Alert**

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

[](https://cdn.guru99.com/images/3-2016/032216_1314_AlertPopuph1.png)

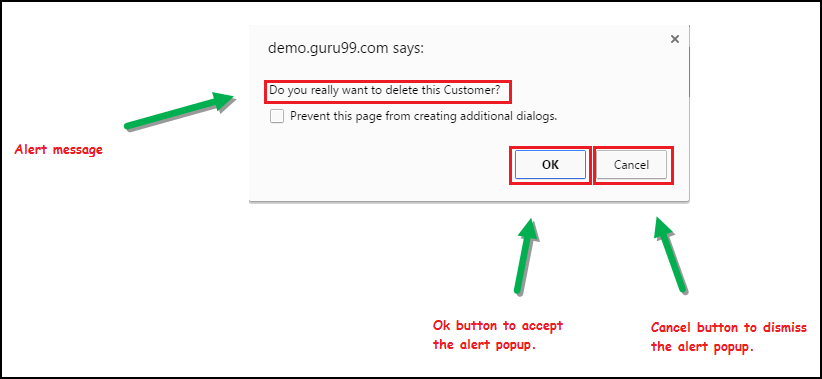
**2) Prompt Alert.**

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

[](https://cdn.guru99.com/images/3-2016/032216_1314_AlertPopuph2.png)

**3) Confirmation Alert.**

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

[](https://cdn.guru99.com/images/3-2016/032216_1314_AlertPopuph3.png)

**How to handle Alert in Selenium WebDriver**

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

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

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

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

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

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

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

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

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

Handling alert in Selenium:

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

alert.dismiss();

alert.accept();

String a=”hello”;

Alert.sendKeys(a);

Handling Multiple Windows:

Synchronization in Selenium:

Implicit

The implicit wait will tell to the web driver to wait for certain amount of time before it throws a "No Such Element Exception". The default setting is 0. Once we set the time, web driver will wait for that time before throwing an exception.

We should note that implicit waits will be in place for the entire time the browser is open. This means that any search for elements on the page could take the time the implicit wait is set for.

## FluentWait Command

**Purpose**: Each **FluentWait** instance defines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition. Furthermore, the user may configure the wait to ignore specific types of exceptions whilst waiting, such as **NoSuchElementExceptions** when searching for an element on the page.

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | // Waiting 30 seconds for an element to be present on the page, checking      // for its presence once every 5 seconds.      Wait wait = new FluentWait(driver)        .withTimeout(30, SECONDS)        .pollingEvery(5, SECONDS)        .ignoring(NoSuchElementException.class);      WebElement foo = wait.until(new Function() {        public WebElement apply(WebDriver driver) {        return driver.findElement(By.id("foo"));        }       }); |

## Explicit Wait

It is more extendible in the means that you can set it up to wait for any condition you might like. Usually, you can use some of the prebuilt **ExpectedConditions** to wait for elements to become clickable, visible, invisible, etc.

|  |  |
| --- | --- |
| 1  2  3 | WebDriverWait wait = new WebDriverWait(driver, 10);    WebElement element = wait.until(ExpectedConditions.elementToBeClickable(By.id("someid |

Java



# Difference Between Implicit, Explicit and Fluent Wait

**Implicit Wait:** During Implicit wait if the Web Driver cannot find it immediately because of its availability, it will keep polling (around 250 milli seconds) the DOM to get the element. If the element is not available within the specified Time an NoSuchElementException will be raised. The default setting is zero. Once we set a time, the Web Driver waits for the period of the WebDriver object instance.

**Explicit Wait:** There can be instance when a particular element takes more than a minute to load. In that case you definitely not like to set a huge time to Implicit wait, as if you do this your browser will going to wait for the same time for every element.

To avoid that situation you can simply put a separate time on the required element only. By following this your browser implicit wait time would be short for every element and it would be large for specific element.

**Fluent Wait:** Let’s say you have an element which sometime appears in just 1 second and some time it takes minutes to appear. In that case it is better to use fluent wait, as this will try to find element again and again until it find it or until the final timer runs out.

;

Multiple Windows in selenium

WebDriver:

Handling Form Elements

Locators:

https://www.softwaretestingclass.com/complete-guide-on-xpath-in-selenium/

### Different types of Locators in Selenium are as follows:

i. [ID](https://www.softwaretestingmaterial.com/how-to-locate-element-by-id-locator/)  
ii. [Name](https://www.softwaretestingmaterial.com/how-to-locate-element-by-name-locator/)  
iii. [Class Name](https://www.softwaretestingmaterial.com/how-to-locate-element-by-class-name-locator/)  
iv. [Tag Name](https://www.softwaretestingmaterial.com/how-to-locate-element-by-tag-name-locator/)  
v. [Link Text & Partial Link Text](https://www.softwaretestingmaterial.com/how-to-locate-element-by-link-text-and-partial-link-text-locator/)  
vi. [CSS Selector](https://www.softwaretestingmaterial.com/css-selector-selenium-webdriver-tutorial/)  
vii. [XPath](https://www.softwaretestingmaterial.com/how-to-locate-element-by-xpath-locator/)

Locating elements in WebDriver is done by using the method “**findElement(By.locator())**“.



|  |  |
| --- | --- |
| 1  2  3 | <td>  <input id="email" class="inputtext" type="email" tabindex="1" value="" name="email">  </td> |

WebDriver driver = new FirefoxDriver();

driver.get("https://www.facebook.com/");

driver.findElement(By.className("inputtext")).sendKeys("Software Testing”);

Tag:

**Tag Name Locator:**

Tag Name locator is used to find the elements matching the specified Tag Name. It is very helpful when we want to extract the content within a Tag.

Css Selector:

### **CSS Selectors Locator:**

There is a debate on the performance of CSS Locator and XPath locator and the debate on the performance of CSS and XPath locator is out of scope of this post. Most of the automation testers believe that using CSS selectors makes the execution of script faster compared to XPath locator. CSS Selector locator is always the best way to locate elements on the page. CSS is always same irrespective of browsers.

Following are some of the mainly used formats of CSS Selectors.

* Tag and ID
* Tag and Class
* Tag and Attribute
* Tag, Class, and Attribute
* Sub-String Matches
  + Starts With (^)
  + Ends With (**$**)
  + Contains (**\***)
* Child Elements
  + Direct Child
  + Sub-child
  + nth-child

<https://www.softwaretestingmaterial.com/css-selector-selenium-webdriver-tutorial/>

XPath Locator:

<https://www.softwaretestingclass.com/using-contains-sibling-ancestor-to-find-element-in-selenium/>

<https://www.guru99.com/xpath-selenium.html>

In Selenium automation, if the elements are not found by the general locators like id, class, name, etc. then XPath is used to find an element on the web page .

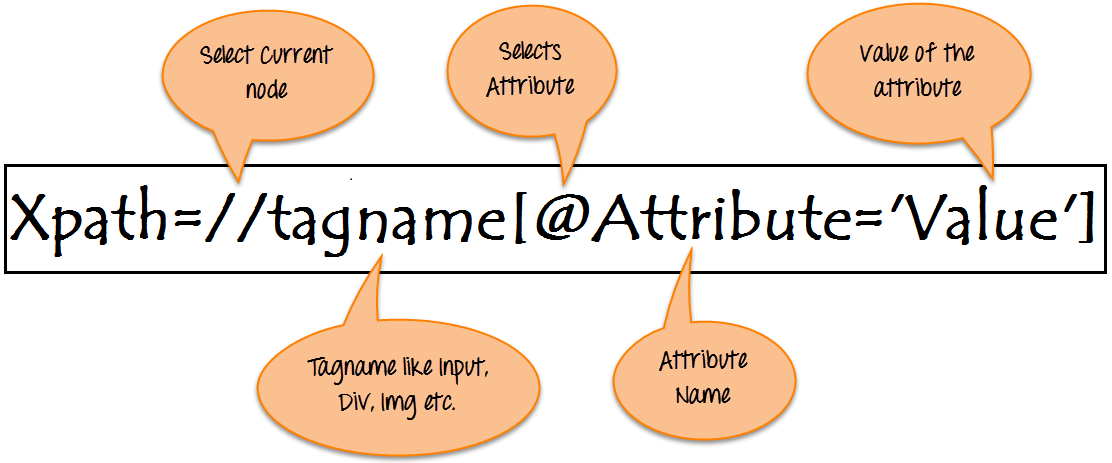
In this tutorial, we will learn about the xpath and different XPath expression to find the complex or dynamic elements, whose attributes changes dynamically on refresh or any operations.

In this tutorial, you will learn-

* [What is XPath](https://www.guru99.com/xpath-selenium.html#1)
* [Types of X-path](https://www.guru99.com/xpath-selenium.html#2)
* [Using XPath Handling complex & Dynamic elements in Selenium](https://www.guru99.com/xpath-selenium.html#3)

## What is XPath

XPath is defined as **XML path**. **It is a syntax or language for finding any element on the web page using XML path expression**. XPath is used to find the location of any element on a webpage using HTML DOM structure. The basic format of XPath is explained below with screen shot.

[](https://www.guru99.com/images/3-2016/032816_0758_XPathinSele1.png)

**Syntax for XPath:**

XPath contains the path of the element situated at the web page. Standard syntax for creating XPath is.

Xpath=//tagname[@attribute='value']

* **// :** Select current node.
* **Tagname:** Tagname of the particular node.
* **@:** Select attribute.
* **Attribute:** Attribute name of the node.
* **Value:** Value of the attribute.

https://www.softwaretestingmaterial.com/dynamic-xpath-in-selenium/

## Types of X-path

There are two types of XPath:

**1) Absolute XPath .**

**2) Relative XPath .**

**Absolute XPath** :

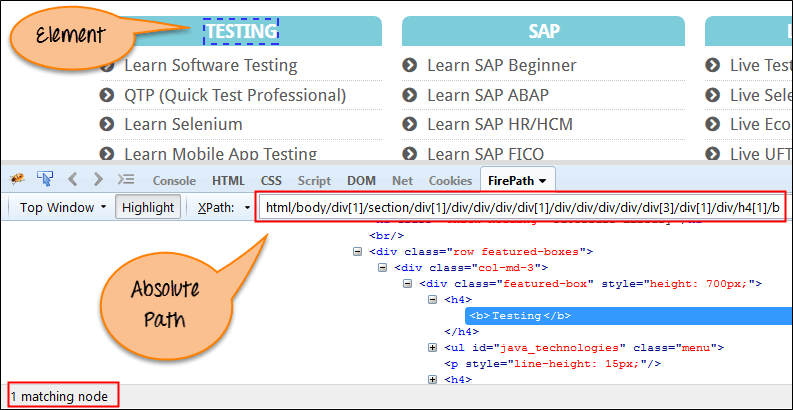
It is the direct way to find the element, but the disadvantage of the absolute XPath is that if there are any changes made in the path of the element then that XPath gets failed.

The key characteristic of XPath is that it begins with the single forward slash(/) ,which means you can select the element from the root node.

Below is the example of an absolute xpath expression of the element shown in the below screen.

**Absolute xpath:**

html/body/div[1]/section/div[1]/div/div/div/div[1]/div/div/div/div/div[3]/div[1]/div/h4[1]/b

[](https://www.guru99.com/images/3-2016/032816_0758_XPathinSele2.png)

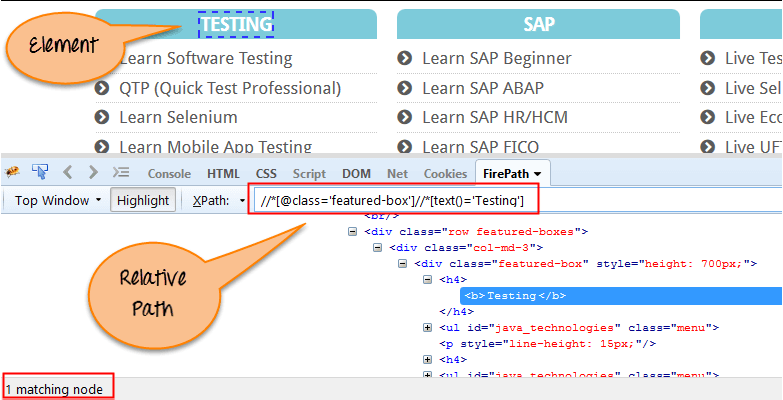
**Relative xpath:**

For Relative Xpath the path starts from the middle of the HTML DOM structure. It starts with the double forward slash (//), which means it can search the element anywhere at the webpage.

You can start from the middle of the HTML DOM structure and no need to write long xpath.

Below is the example of a relative XPath expression of the same element shown in the below screen. This is the common format used to find element through a relative XPath.

Relative xpath: //\*[@class='featured-box']//\*[text()='Testing']

[](https://www.guru99.com/images/3-2016/032816_0758_XPathinSele3.png)

**What are XPath axes.**

Handling WebTable:

<https://www.guru99.com/handling-dynamic-selenium-webdriver.html>

<https://www.guru99.com/xpath-selenium.html>

FindElements and FindElement:

<http://toolsqa.com/selenium-webdriver/findelement-and-findelements-command/>

## Difference between FindElement & FindElements Commands

The difference between ***findElement()*** and ***findElements()*** method is the first returns a WebElement object otherwise it throws an exception and the latter returns a List of WebElements, it can return an empty list if no DOM elements match the query.

***findElement()***

* On Zero Match : throws NoSuchElementException
* On One Match : returns WebElement
* On One+ Match : returns the first appearance in DOM

***findElements()***

* On Zero Match : return an empty list
* On One Match : returns list of one WebElement only
* On One+ Match : returns list with all matching instance

Alert & Popup Window Handling in Selenium WebDriver

In this tutorial, we will learn about different types of alert found in web application [Testing](https://www.guru99.com/software-testing.html) and how to handle Alert in Selenium WebDriver. We will also see how do we accept and reject the alert depending upon the alert types.

In this tutorial, you will learn-

[What is Alert?](https://www.guru99.com/alert-popup-handling-selenium.html#1)

[How to handle Alert in Selenium WebDriver](https://www.guru99.com/alert-popup-handling-selenium.html#2)

[How to handle Selenium Popup window using Webdriver](https://www.guru99.com/alert-popup-handling-selenium.html#3)

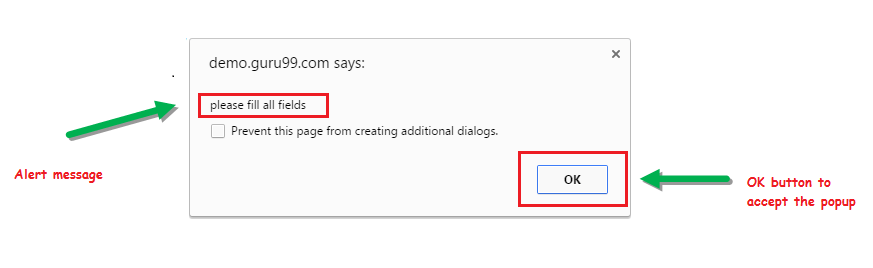
What is Alert?

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

Here are few alert types:

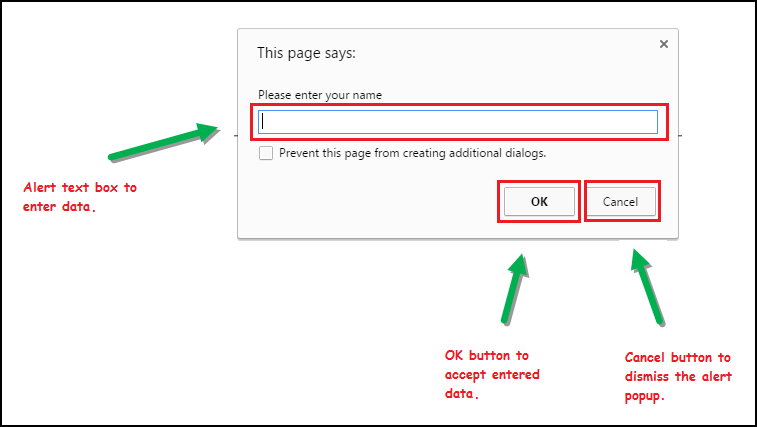
1) Simple Alert

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

[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph1.png)

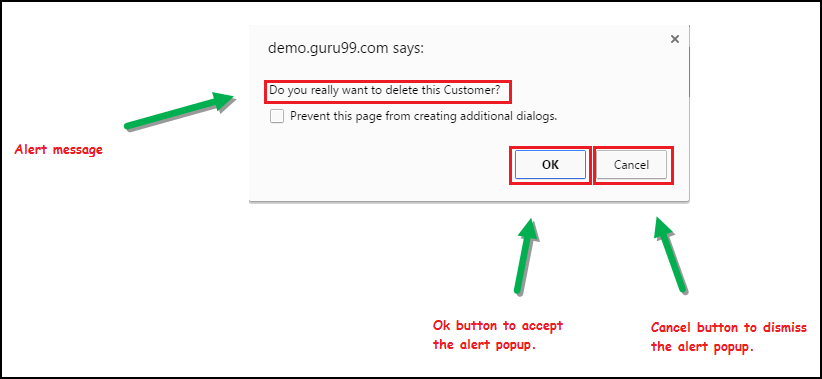
2) Prompt Alert.

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

[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph2.png)

3) Confirmation Alert.

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

[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph3.png)

How to handle Alert in Selenium WebDriver

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

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

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

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

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

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

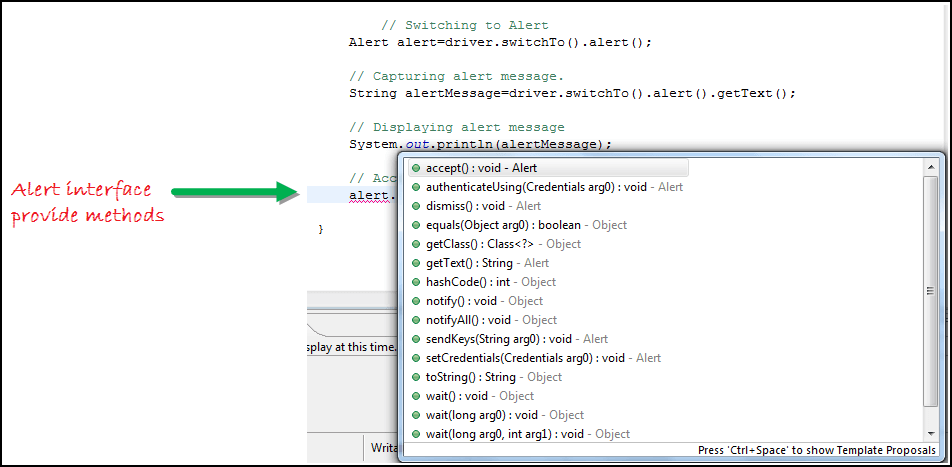
driver.switchTo().alert().getText();

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

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

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

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

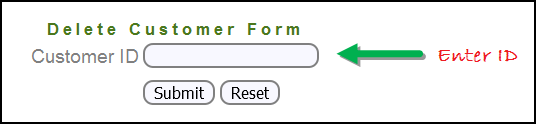
[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph4.png)

Now we automate the given below scenario.

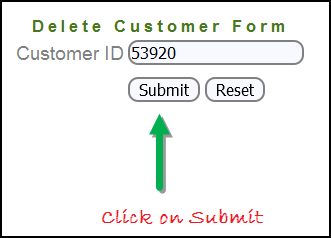
In this scenario, we will use Guru99 demo site to illustrate Selenium Alert handling.

Step 1) Launch the web browser and open the site "<http://demo.guru99.com/test/delete_customer.php> "

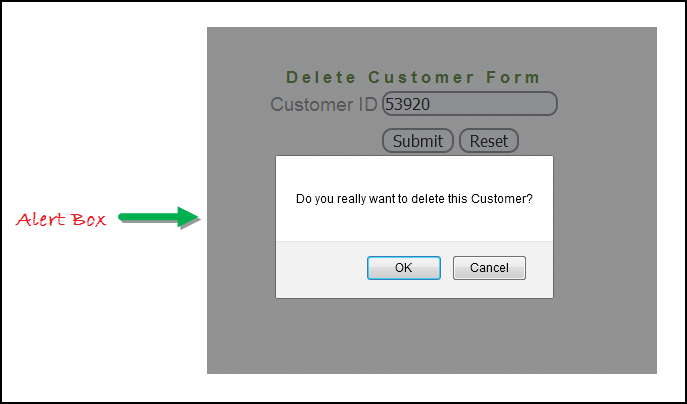
Step 2) Enter Any Customer id.

[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph8.png)

Step 3) After entering the customer ID, Click on the "Submit" button.

[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph9.png)

Step 4) Reject/accept the alert.

[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph10.png)

Handling Alert in Selenium Webdriver using above scenario

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.NoAlertPresentException;

import org.openqa.selenium.Alert;

public class AlertDemo {

public static void main(String[] args) throws NoAlertPresentException,InterruptedException {

System.setProperty("webdriver.chrome.driver","G:\\chromedriver.exe");

WebDriver driver = new ChromeDriver();

// Alert Message handling

driver.get("http://demo.guru99.com/test/delete\_customer.php");

driver.findElement(By.name("cusid")).sendKeys("53920");

driver.findElement(By.name("submit")).submit();

// Switching to Alert

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

// Capturing alert message.

String alertMessage= driver.switchTo().alert().getText();

// Displaying alert message

System.out.println(alertMessage);

Thread.sleep(5000);

// Accepting alert

alert.accept();

}

}

Output :

When you execute the above code, it launches the site. Try to delete Customer ID by handling confirmation alert that displays on the screen, and thereby deleting customer id from the application.

How to handle Selenium Pop-up window using Webdriver

**Handling multiple windows:**

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

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

Driver.getWindowHandles();

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

Driver.getWindowHandle();

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

To handle multiple windows in Selenium WebDriver, We follow the following steps.

Now, we will automate the given below scenario to see how to handle multiple windows using Selenium Webdriver.

In this scenario, we will use "Guru99" demo site to illustrate window handling.

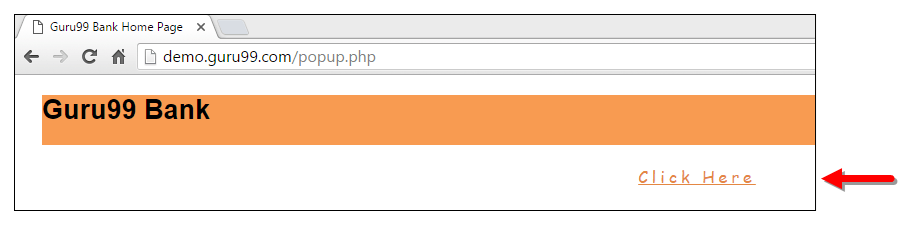
Step 1) Launch the site.

Launch the browser and open the site " http://demo.guru99.com/popup.php "

[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph11.png)

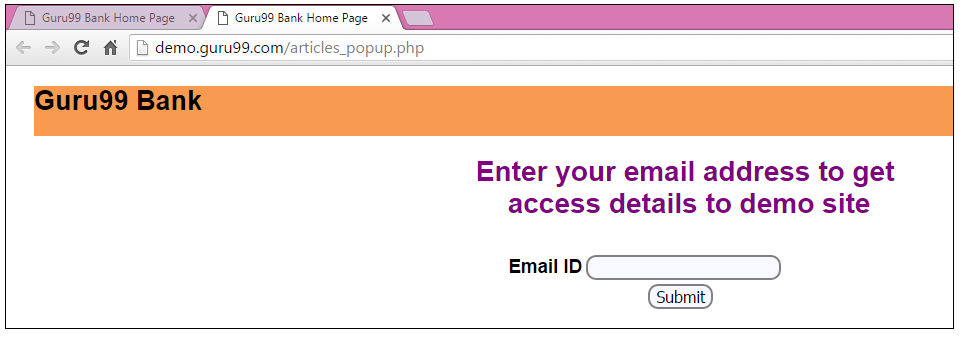
Step 2) Click on link "Click Here ".

When the user clicks on the " Click Here " link, new child window opens.

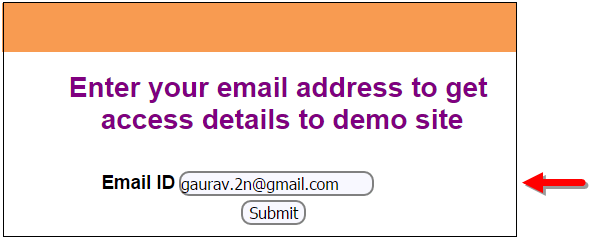
[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph12.png)

Step 3) New Child Window opens.

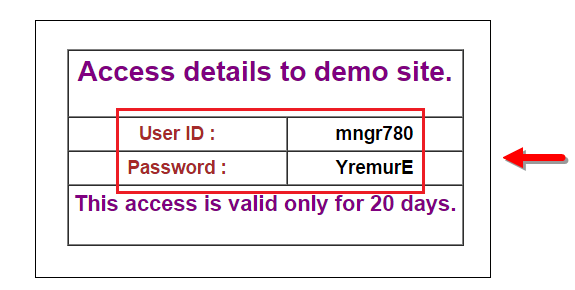
A new window opens, ask the user to enter email id and submit the page.

[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph13.png)

Step 4) Enter your email ID and submit.

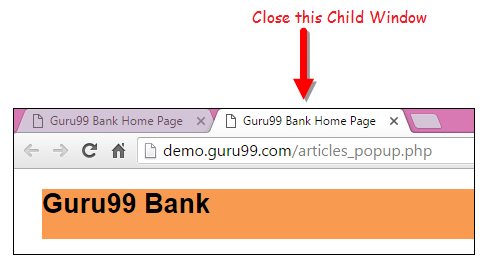
[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph14.png)

Step 5) Display the Access Credentials on submitting the page.

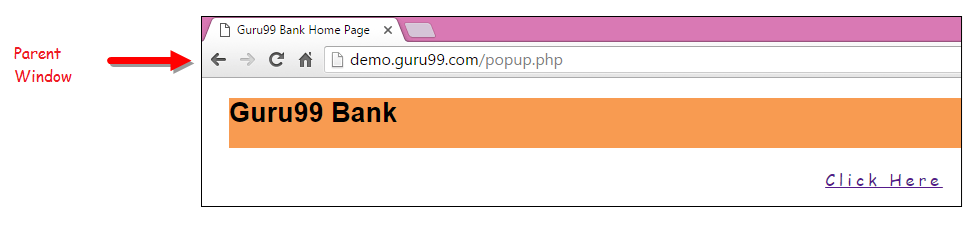
[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph15.png)

When you execute the code, you will see the child window is open in new tab.

Close the Child window on which credentials are displayed.

[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph16.png)

Switch to the parent window.

[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph17.png)

Handling multiple windows in selenium webdriver using above scenario.

import java.util.Iterator;

import java.util.Set;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class WindowHandle\_Demo {

public static void main(String[] args) throws InterruptedException {

WebDriver driver=new FirefoxDriver();

//Launching the site.

driver.get("http://demo.guru99.com/popup.php");

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

driver.findElement(By.xpath("//\*[contains(@href,'popup.php')]")).click();

String MainWindow=driver.getWindowHandle();

// To handle all new opened window.

Set<String> s1=driver.getWindowHandles();

Iterator<String> i1=s1.iterator();

while(i1.hasNext())

{

String ChildWindow=i1.next();

if(!MainWindow.equalsIgnoreCase(ChildWindow))

{

// Switching to Child window

driver.switchTo().window(ChildWindow);

driver.findElement(By.name("emailid"))

.sendKeys("[gaurav.3n@gmail.com](mailto:gaurav.3n@gmail.com)

");

driver.findElement(By.name("btnLogin")).click();

// Closing the Child Window.

driver.close();

}

}

// Switching to Parent window i.e Main Window.

driver.switchTo().window(MainWindow);

}

}

Output:

When you execute the above code, it launches the site and on clicking the link "Click here," it opens up a child window in a new tab. You can close the child window, and switch to the parent window once the operation is completely done. Hence handling more than one window in the application.

**Code to open in new window**

package com.frame;

import org.openqa.selenium.Alert;

import org.openqa.selenium.By;

import org.openqa.selenium.JavascriptExecutor;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.ie.InternetExplorerDriver;

public class IFrameDemo {

public static void main(String[] args) {

// TODO Auto-generated method stub

System.setProperty("webdriver.chrome.driver", "G:\\Softwares\\chromedriver\_win32 (1)\\chromedriver.exe");

WebDriver driver=new ChromeDriver();

driver.get("file:///D:/SeleniumCEP/DemoHTML/iFrameDemo.html");

driver.switchTo().frame("IF2"); //switching the frame by ID

//driver.switchTo().frame(1); //switching the frame by ID

driver.findElement(By.name("uname")).sendKeys("Manisha");

driver.findElement(By.name("pwd")).sendKeys("Syntel123$");

driver.findElement(By.name("pwd")).click();

driver.findElement(By.name("btnSbt")).click();

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

try {

Thread.sleep(5000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

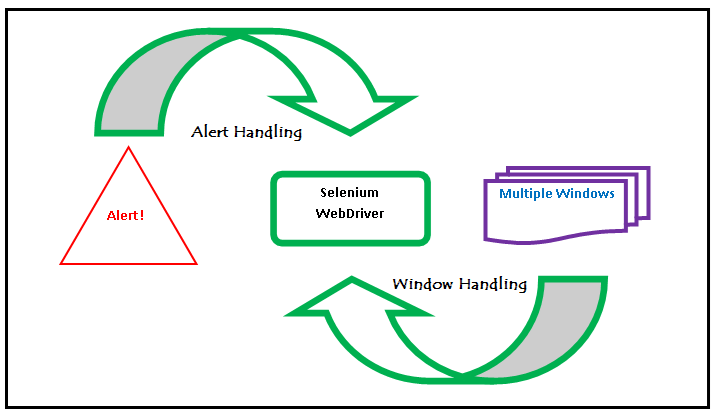
alert.accept();

String a = "window.open('D://SeleniumCEP//DemoHTML//success.html','\_blank');";

((JavascriptExecutor)driver).executeScript(a);

}

}

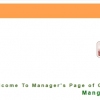
[](https://www.guru99.com/images/3-2016/032216_1314_AlertPopuph18.png)

[Verify Tooltip Using Selenium WebDriver](https://www.guru99.com/verify-tooltip-selenium-webdriver.html)

The tooltip is a text that appears when a mouse hovers over an object like a link, an image, a...

[Read more](https://www.guru99.com/verify-tooltip-selenium-webdriver.html)

Selenium

[](https://www.guru99.com/introduction-testng-groups.html)

[Introduction to TestNG Groups](https://www.guru99.com/introduction-testng-groups.html)

TestNG is a Testing framework that covers different types of test designs like unit, functional,...

[Read more](https://www.guru99.com/introduction-testng-groups.html)

Selenium

[](https://www.guru99.com/desired-capabilities-selenium.html)

[Desired Capabilities in Selenium WebDriver](https://www.guru99.com/desired-capabilities-selenium.html)

Action Classes:

**Keyboard & Mouse Event using Action Class in Selenium Webdriver**

**Details**

Last Updated: 12 November 2018

In this tutorial, we will learn handling Keyboard and Mouse Event in Selenium Webdriver

**Handling Keyboard & Mouse Events**

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

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

<https://www.softwaretestingmaterial.com/keyboard-mouse-events-using-selenium-actions-class/>

Synchronization:

## Implicit Wait

Selenium WebDriver has borrowed the idea of **implicit waits** from **Watir**. This means that we can tell Selenium that we would like it to wait for a certain amount of time before throwing an **exception** that it cannot find the element on the page. We should note that implicit waits will be in place for the entire time the browser is open. This means that any search for elements on the page could take the time the implicit wait is set for.

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | WebDriver driver = new FirefoxDriver();    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);    driver.get("http://url\_that\_delays\_loading");    WebElement myDynamicElement = driver.findElement(By.id("myDynamicElement")); |

## Fluent Wait

Each **FluentWait** instance defines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition. Furthermore, the user may configure the wait to ignore specific types of exceptions whilst waiting, such as **NoSuchElementExceptions** when searching for an element on the page.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | // Waiting 30 seconds for an element to be present on the page, checking      // for its presence once every 5 seconds.      Wait wait = new FluentWait(driver)        .withTimeout(30, SECONDS)        .pollingEvery(5, SECONDS)        .ignoring(NoSuchElementException.class);      WebElement foo = wait.until(new Function() {        public WebElement apply(WebDriver driver) {        return driver.findElement(By.id("foo"));        }       }); |

## Explicit Wait

It is more extendible in the means that you can set it up to wait for any condition you might like. Usually, you can use some of the prebuilt **ExpectedConditions** to wait for elements to become clickable, visible, invisible, etc.

Java



|  |  |
| --- | --- |
| 1  2  3 | WebDriverWait wait = new WebDriverWait(driver, 10);    WebElement element = wait.until(ExpectedConditions.elementToBeClickable(By.id("someid"))); |

# Difference Between Implicit, Explicit and Fluent Wait

**Implicit Wait:** During Implicit wait if the Web Driver cannot find it immediately because of its availability, it will keep polling (around 250 milli seconds) the DOM to get the element. If the element is not available within the specified Time an NoSuchElementException will be raised. The default setting is zero. Once we set a time, the Web Driver waits for the period of the WebDriver object instance.

**Explicit Wait:** There can be instance when a particular element takes more than a minute to load. In that case you definitely not like to set a huge time to Implicit wait, as if you do this your browser will going to wait for the same time for every element.

To avoid that situation you can simply put a separate time on the required element only. By following this your browser implicit wait time would be short for every element and it would be large for specific element.

**Fluent Wait:** Let’s say you have an element which sometime appears in just 1 second and some time it takes minutes to appear. In that case it is better to use fluent wait, as this will try to find element again and again until it find it or until the final timer runs out.

**Solutions:** We always get confuse when it comes to using Wait commands, to better understand it we need to remember that there is a difference between several scenarios:

An element not being present at all in the DOM.

An element being present in the DOM but not visible.

An element being present in the DOM but not enabled. (i.e. clickable)

There are pages which get displayed with the **JavaScript**, the elements are already present in the browser **DOM**, but are not visible. The implicit wait only waits for an element to appear in the DOM, so it returns immediately, but when you try to interact with the element you get a **NoSuchElementException**. You could test this hypothesis by writing a helper method that explicit wait for an element to be visible or clickable.

V

Handling iFrame in selenium:

Before starting we have to understand that to work with different iFrames on a page we have to switch between these iFrames. To Switch between iFrames we have to use the driver’s ***switchTo().frame*** command. We can use the *switchTo().frame()* in three ways:

* ***switchTo.frame(int frameNumber)*:** Pass the frame index and driver will switch to that frame.
* ***switchTo.frame(string frameNameOrId)*:** Pass the frame element Name or ID and driver will switch to that frame.
* ***switchTo.frame(WebElement frameElement)*:** Pass the frame web element and driver will switch to that frame.

ScreenShot in selenium

* Taking Screenshot in Selenium is a 3 Step process
* **Step 1)** Convert web driver object to TakeScreenshot
* TakesScreenshot scrShot =((TakesScreenshot)webdriver);
* **Step 2)** Call getScreenshotAs method to create image file
* File SrcFile=scrShot.getScreenshotAs(OutputType.FILE);
* **Step 3)** Copy file to Desired Location

contains-sbiling-ancestor-to-find-element-in-selenium

<https://www.guru99.com/using-contains-sbiling-ancestor-to-find-element-in-selenium.html>

If a simple [XPath](https://www.guru99.com/xpath-selenium.html) is not able to find a complicated web element for our test script, we need to use the functions from XPath 1.0 library. With the combination of these functions, we can create more specific XPath. Let's discuss a 3 such functions –

1. Contains
2. Sibling
3. Ancestor

Let's study them in detail -

**Contains**: By using 'contains' function in XPath, we can extract all the elements which matches a particular text value.

**Sibling**: Using sibling keyword, we can fetch a web element on the which is related to some other element.

**Ancestor**: To find an element on the basis of the parent element we can use ancestor attribute of XPath.

https://www.guru99.com/xpath-selenium.html

Uploading and downloading a file selenium:

**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.**

<https://www.guru99.com/upload-download-file-selenium-webdriver.html>

Object Repository:

## What is an Object Repository?

An object repository is a common storage location for all objects. In Selenium WebDriver context, objects would typically be the locators used to uniquely identify web elements.

The major advantage of using object repository is the segregation of objects from test cases. If the locator value of one webelement changes, only the object repository needs to be changed rather than making changes in all test cases in which the locator has been used. Maintaining an object repository increases the modularity of framework implementation.

In this tutorial, you will learn-

* [What is an Object Repository?](https://www.guru99.com/object-repository-selenium.html#1)
* [Types of Object Repositories in Selenium Web Driver](https://www.guru99.com/object-repository-selenium.html#2)
* [Selenium Web Driver Object repository using Properties file](https://www.guru99.com/object-repository-selenium.html#3)
* [Selenium WebDriver Object Repository Using XML File](https://www.guru99.com/object-repository-selenium.html#4)

## Types of Object Repositories in Selenium Web Driver

Selenium WebDriver does not offer an in-built object repository by default. However, object repositories can be built using the key-value pair approach wherein the key refers to the name given to the object and value refers to the properties used to uniquely identify an object within the web page.

The following are the types of object repositories that can be created in Selenium WebDriver.

1. Object Repository using Properties file
2. Object Repository using XML file

Demo: D:\ManishaTesting\Testing\AdvSelenium\ObjectRepository

POM (Page Object model):

## What is Page Object Model?

Page Object Model is a design pattern to create **Object Repository** for web UI elements. Under this model, for each web page in the application, there should be corresponding page class. This Page class will find the WebElements of that web page and also contains Page methods which perform operations on those WebElements.

Name of these methods should be given as per the task they are performing, i.e., if a loader is waiting for the payment gateway to appear, POM method name can be waitForPaymentScreenDisplay().

https://www.guru99.com/page-object-model-pom-page-factory-in-selenium-ultimate-guide.html

D:\ManishaTesting\Testing\AdvSelenium\POMDemo

DataBaseTesting :

**Database Testing using Selenium: Step by Step Guide**

**Details**

Last Updated: 13 December 2018

Selenium Webdriver is limited to [Testing](https://www.guru99.com/software-testing.html) your applications using Browser. To use Selenium Webdriver for Database Verification you need to use the JDBC ("Java Database Connectivity").

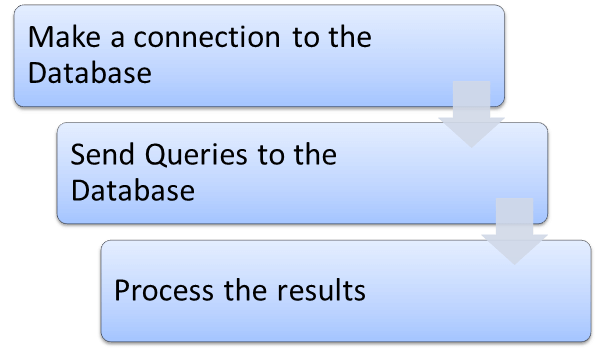
JDBC (Java Database Connectivity) is a [SQL](https://www.guru99.com/sql.html) level API that allows you to execute SQL statements. It is responsible for the connectivity between the [Java](https://www.guru99.com/java-tutorial.html) Programming language and a wide range of databases. The JDBC API provides the following classes and interfaces

* Driver Manager
* Driver
* Connection
* Statement
* ResultSet
* SQLException

In this tutorial, you will learn

* [Make a connection to the Database](https://www.guru99.com/database-testing-using-selenium-step-by-step-guide.html#1)
* [Send Queries to the Database](https://www.guru99.com/database-testing-using-selenium-step-by-step-guide.html#2)
* [Process the results](https://www.guru99.com/database-testing-using-selenium-step-by-step-guide.html#3)
* [Example of Database Testing with Selenium](https://www.guru99.com/database-testing-using-selenium-step-by-step-guide.html#4)

In order to test your Database using Selenium, you need to observe the following 3 steps

[](https://www.guru99.com/images/6-2015/052615_1013_DatabaseTes1.png)

## 1) Make a connection to the Database

In order to make a connection to the database the syntax is

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

Here,

* Userid is the username configured in the database
* Password of the configured user
* URL is of format jdbc:< dbtype>://ipaddress:portnumber/db\_name"
* <dbtype>- The driver for the database you are trying to connect. To connect to oracle database this value will be "oracle"

For connecting to database with name "emp" in MYSQL URL will bejdbc:mysql://localhost:3036/emp

And the code to create connection looks like

Connection con = DriverManager.getConnection(dbUrl,username,password);

You also need to load the JDBC Driver using the code

<https://www.seleniumeasy.com/selenium-tutorials/how-to-run-webdriver-in-ie-browser>

https://www.guru99.com/using-robot-api-selenium.html

Class.forName("com.mysql.jdbc.Driver");

## 2) Send Queries to the Database

Once connection is made, you need to execute queries.

You can use the Statement Object to send queries.

Statement stmt = con.createStatement();

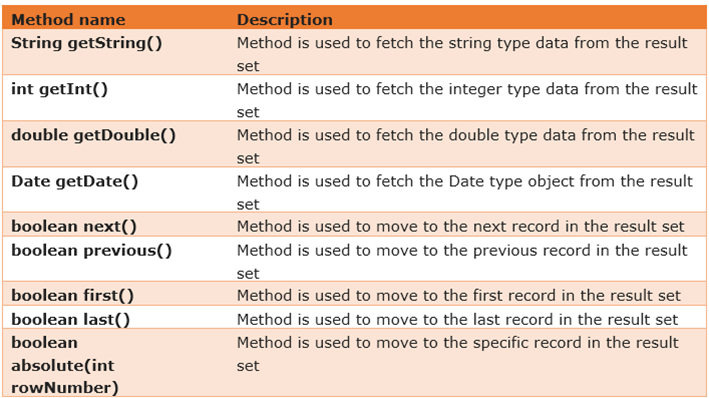
Once the statement object is created use the executeQuery method to execute the SQL queries

stmt.executeQuery(select \* from employee;);

## 3) Process the results

Results from the executed query are stored in the ResultSet Object.

Java provides loads of advance methods to process the results. Few of the methods are listed below

[](https://www.guru99.com/images/6-2015/052615_1013_DatabaseTes2.png)

## Example of Database Testing with Selenium

**Step 1)** Install [MySQL Server](http://dev.mysql.com/downloads/mysql/) and [MySQL Workbench](http://dev.mysql.com/downloads/workbench/)

Check out the complete guide to Mysql & Mysql Workbench [here](https://www.guru99.com/sql.html)

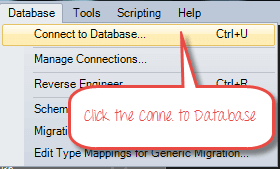
While installing MySQL Server, please note the database

* Username
* Password
* Port Number

It will be required in further steps.

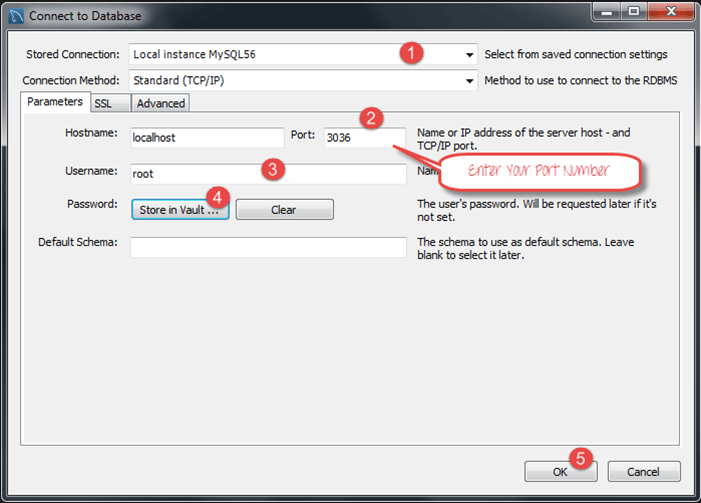
MySQL Workbench makes it easy to administer the database without the need to code SQL. Though, you can also use the MySQL Terminal to interact with the database.

**Step 2)** In MySQL WorkBench, connect to your MySQL Server

[](https://www.guru99.com/images/6-2015/052615_1013_DatabaseTes3.png)

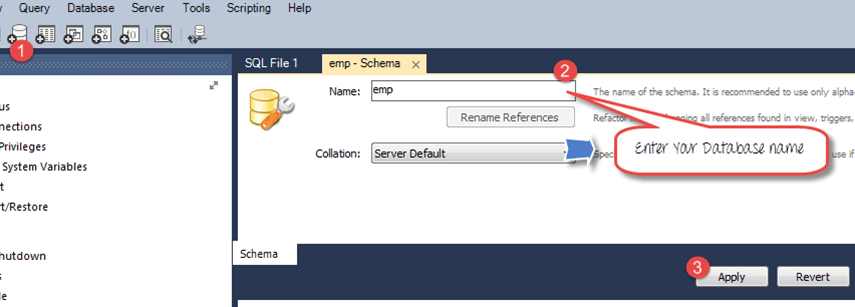
In the next screen,

1. Select Local Instance of MySQL
2. Enter Port Number
3. Enter Username
4. Enter Password
5. Click OK

[](https://www.guru99.com/images/6-2015/052615_1013_DatabaseTes4.png)

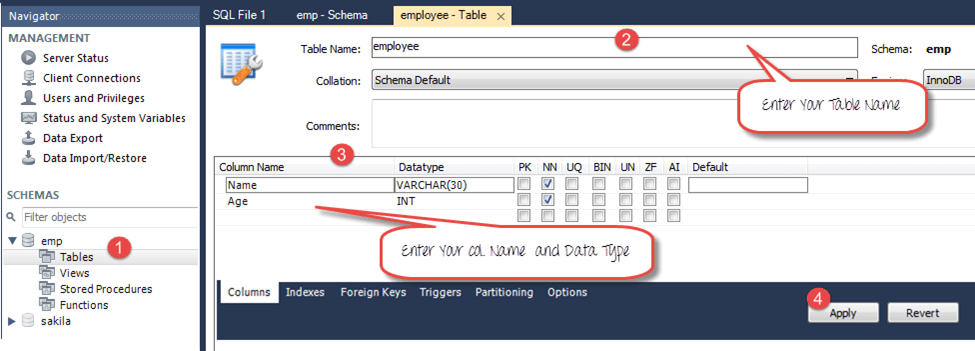
**Step 3)** To Create Database,

1. Click create Schema Button
2. Enter Name of Schema/Database
3. Click Apply

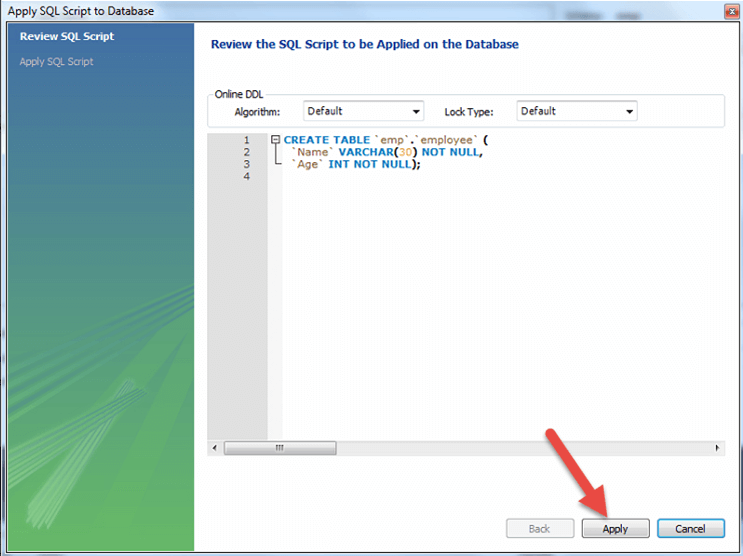
[](https://www.guru99.com/images/6-2015/052615_1013_DatabaseTes5.png)

**Step 4)** In the navigator menu,

1. Click on Tables, beneath the emp database
2. Enter Table name as employee
3. Enter Fields as Name and Age
4. Click Apply

[](https://www.guru99.com/images/6-2015/052615_1013_DatabaseTes6.png)

You will see the following pop-up. Click Apply

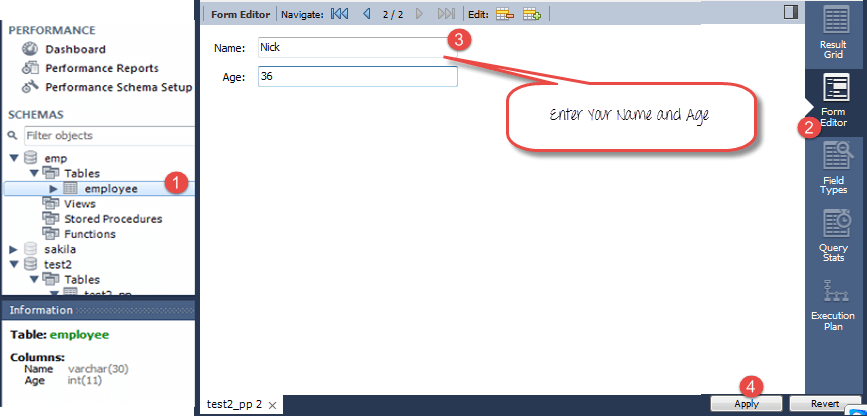
[](https://www.guru99.com/images/6-2015/052615_1013_DatabaseTes7.png)

**Step 5)** We will create following data

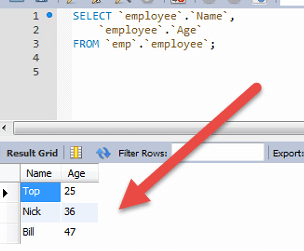
|  |  |
| --- | --- |
| **Name** | **Age** |
| Top | 25 |
| Nick | 36 |
| Bill | 47 |

To create data into the Table

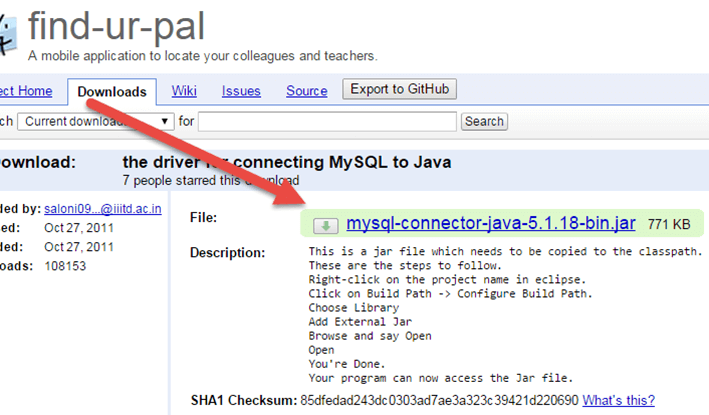
1. In navigator, select the employee table
2. In right pane, click Form Editor
3. Enter Name and Age
4. Click Apply

[](https://www.guru99.com/images/6-2015/052615_1013_DatabaseTes8.png)

Repeat the process until all data is created

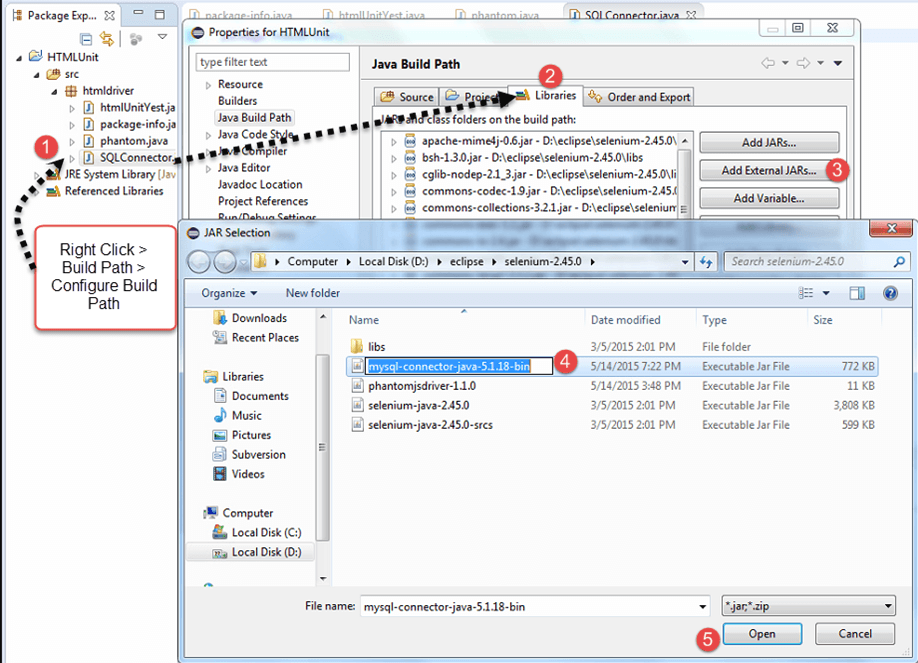
[](https://www.guru99.com/images/6-2015/052615_1013_DatabaseTes9.png)

**Step 6)** Download the MySQL JDBC connector [here](https://code.google.com/p/find-ur-pal/downloads/detail?name=mysql-connector-java-5.1.18-bin.jar&)

[](https://www.guru99.com/images/6-2015/052615_1013_DatabaseTes10.png)

**Step 7)** Add the downloaded Jar to your Project

1. Right click on your Java File. Then click on Build Pathà Configure build path
2. Select the libraries
3. Click on add external JARs
4. You can see MySQL connector java in your library
5. Click on open to add it to the project

[](https://www.guru99.com/images/6-2015/052615_1013_DatabaseTes11.png)

**Step 8)** Copy the following code into the editor

Package htmldriver;

import java.sql.Connection;

import java.sql.Statement;

import java.sql.ResultSet;

import java.sql.DriverManager;

import java.sql.SQLException;

public class SQLConnector {

public static void main(String[] args) throws ClassNotFoundException, SQLException {

//Connection URL Syntax: "jdbc:mysql://ipaddress:portnumber/db\_name"

String dbUrl = "jdbc:mysql://localhost:3036/emp";

//Database Username

String username = "root";

//Database Password

String password = "guru99";

//Query to Execute

String query = "select \* from employee;";

//Load mysql jdbc driver

Class.forName("com.mysql.jdbc.Driver");

//Create Connection to DB

Connection con = DriverManager.getConnection(dbUrl,username,password);

//Create Statement Object

Statement stmt = con.createStatement();

// Execute the SQL Query. Store results in ResultSet

ResultSet rs= stmt.executeQuery(query);

// While Loop to iterate through all data and print results

while (rs.next()){

String myName = rs.getString(1);

String myAge = rs.getString(2);

System. out.println(myName+" "+myAge);

}

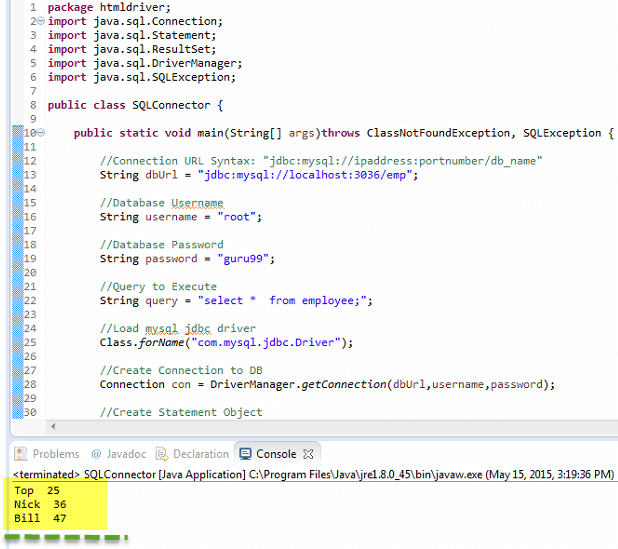
// closing DB Connection

con.close();

}

}

**Step 8)** Execute the code, and check the output

[](https://www.guru99.com/images/6-2015/052615_1013_DatabaseTes12.png)

## Summary of Steps for Selenium Database Testing

**Step 1)** Make a connection to the Database using method.

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

**Step 2)** Create Query to the Database using the Statement Object.

Statement stmt = con.createStatement();

**Step 3)** Send the query to database using execute query and store the results in the ResultSet object.

ResultSet rs = stmt.executeQuery(select \* from employee;);

Java provides lots of built-in methods to process the> SQL Output using the ResultSet Object

D:\ManishaTesting\Testing\AdvSelenium\DataBaseTesting

**FrameWorks in selenium:**

DataDriven Framework with Apache POI

allows you to automatically run a test case multiple times with different input and validation values.

<https://www.guru99.com/all-about-excel-in-selenium-poi-jxl.html>

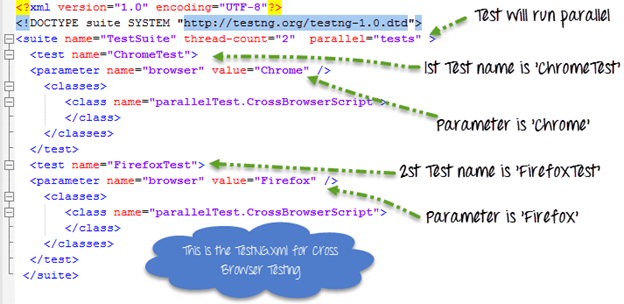
D:\ManishaTesting\Testing\AdvSelenium\DataDrivenPOI

Keyword Driven Frame work:

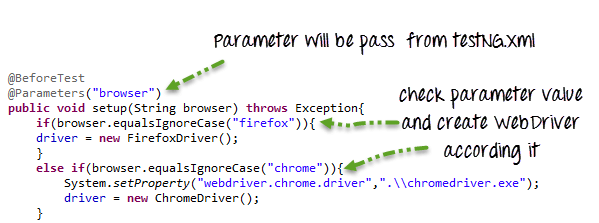
D:\ManishaTesting\Testing\AdvSelenium\KeyWordDriven

Cross Browser Testinh:

Your testing.xml will look like that,

[](https://www.guru99.com/images/AdvanceSelenium/071514_0700_CrossBrowse3.png)

This testing.xml will map with the [Test Case](https://www.guru99.com/test-case.html) which will look like that

[](https://www.guru99.com/images/AdvanceSelenium/071514_0700_CrossBrowse4.png)

Here because the testing.xml has two Test tags ('ChromeTest','FirefoxTest'),this test case will execute two times for 2 different browsers.

First Test 'ChromeTest' will pass the value of parameter 'browser' as 'chrome' so ChromeDriver will be executed. This test case will run on Chrome browser.

Second Test 'FirefoxTest' will pass the value of parameter 'browser' as 'Firefox' so FirefoxDriver will be executed. This test case will run on FireFox browser.

Complete Code:

**Guru99CrossBrowserScript.java**

package parallelTest;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.edge.EdgeDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Parameters;

import org.testng.annotations.Test;

public class CrossBrowserScript {

WebDriver driver;

/\*\*

\* This function will execute before each Test tag in testng.xml

\* @param browser

\* @throws Exception

\*/

@BeforeTest

@Parameters("browser")

public void setup(String browser) throws Exception{

//Check if parameter passed from TestNG is 'firefox'

if(browser.equalsIgnoreCase("firefox")){

//create firefox instance

System.setProperty("webdriver.firefox.marionette", ".\\geckodriver.exe");

driver = new FirefoxDriver();

}

//Check if parameter passed as 'chrome'

else if(browser.equalsIgnoreCase("chrome")){

//set path to chromedriver.exe

System.setProperty("webdriver.chrome.driver",".\\chromedriver.exe");

//create chrome instance

driver = new ChromeDriver();

}

//Check if parameter passed as 'Edge'

else if(browser.equalsIgnoreCase("Edge")){

//set path to Edge.exe

System.setProperty("webdriver.edge.driver",".\\MicrosoftWebDriver.exe");

//create Edge instance

driver = new EdgeDriver();

}

else{

//If no browser passed throw exception

throw new Exception("Browser is not correct");

}

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

}

@Test

public void testParameterWithXML() throws InterruptedException{

driver.get("http://demo.guru99.com/V4/");

//Find user name

WebElement userName = driver.findElement(By.name("uid"));

//Fill user name

userName.sendKeys("guru99");

//Find password

WebElement password = driver.findElement(By.name("password"));

//Fill password

password.sendKeys("guru99");

}

}

**testing.xml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="TestSuite" thread-count="2" parallel="tests" >

<test name="ChromeTest">

<parameter name="browser" value="Chrome" />

<classes>

<class name="parallelTest.CrossBrowserScript">

</class>

</classes>

</test>

<test name="FirefoxTest">

<parameter name="browser" value="Firefox" />

<classes>

<class name="parallelTest.CrossBrowserScript">

</class>

</classes>

</test>

<test name="EdgeTest">

<parameter name="browser" value="Edge" />

<classes>

<class name="parallelTest.CrossBrowserScript">

</class>

</classes>

</test>

</suite>

**NOTE:** To run the test, Right click on the **testing.xml,** Select Run As, and Click TestNG

<https://www.guru99.com/cross-browser-testing-using-selenium.html>

Log4j:

https://www.softwaretestinghelp.com/log4j-tutorial-selenium-tutorial-26/