|  |  |
| --- | --- |
| Purpose | Appium Setup |
| Created By | Abhilasha pandey |
| Created On | 16 October 2015 |
| Reviewed By | Kshitij |
| Version | v1.0 |

**APPIUM**

APPIUM is an open source mobile functional automation testing framework. Appium allows native, hybrid and web application testing and supports automation test on physical devices as well as on emulator or simulator both. It offers cross-platform functional application testing i.e. single API works for both Android and iOS platform test scripts. It has **no** dependency on Mobile device OS. Because, APPIUM has framework or wrapper that translate Selenium Webdriver commands into UIAutomation (iOS) or UIAutomator (Android) commands depending on the device type not any OS type.

Appium supports all languages that have Selenium client libraries like- Java, Objective-C, JavanScript with node.js, PHP, Ruby, Python, C# etc.

Appium is built on the idea that testing native apps shouldn't require including an SDK or recompiling your app. And that you should be able to use your preferred test practices, frameworks, and tools. Appium is an open source project and has made design and tool decisions to encourage a vibrant contributing community.

**SYSTEM REQUIREMENTS**

* Windows

1. Mac OSX 10.7+ or Windows 7+ or Linux
2. Android SDK >= 16 ( Android SDK < 16 in Selendroid mode)

* iOS

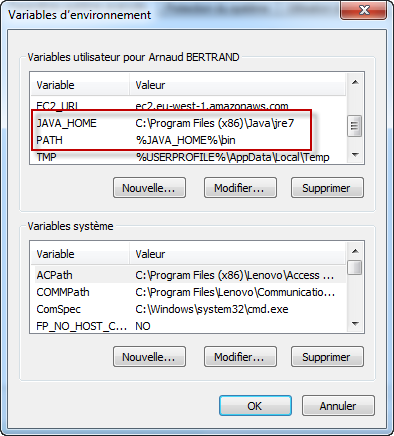
1. Mac OSX 10.7+
2. XCode 4.5+ w/ Command Line Tools

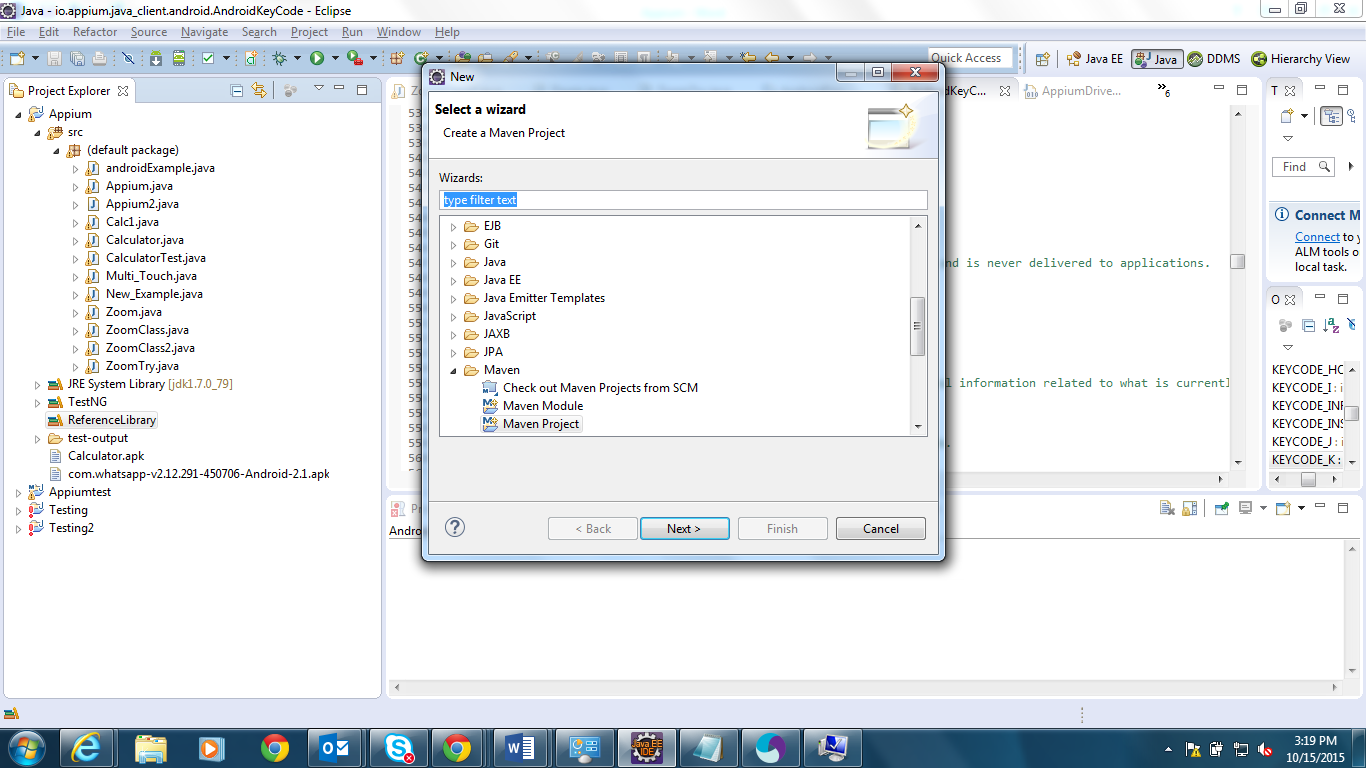
**APPIUM SETUP**

* **Install JDK and configure environment variables**.

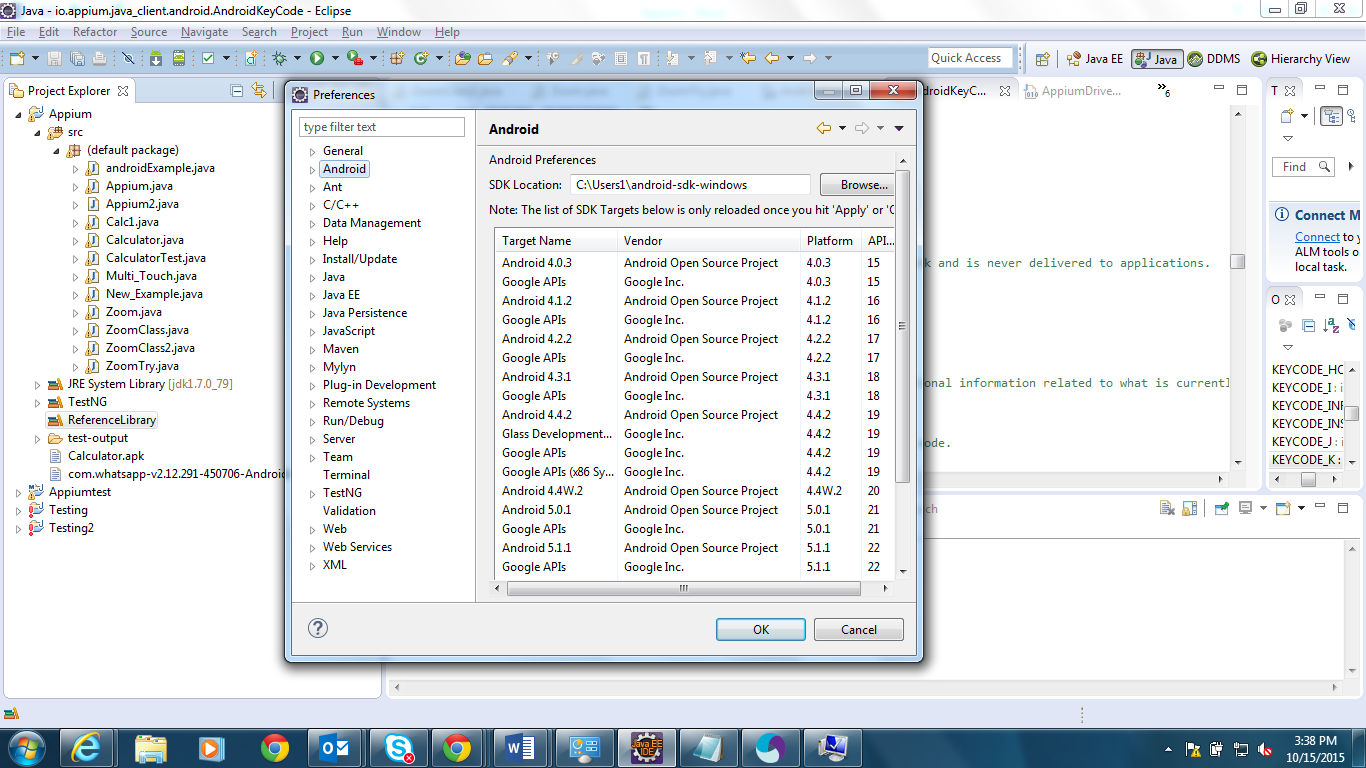
To install JDK, on google type download JDK from Oracle’s website. Download accordingly for 32 bit and 64 bit machine.

(URL: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>)

1. Then go to Start, right click on computer and select ‘Properties’. In the new window select ‘Advanced system setting’ and then click on ‘Environment Variables’.
2. Create a new system variable and name it as JAVA\_HOME. Give the variable value as the path of JDK.
3. Download Eclipse Indigo or above.

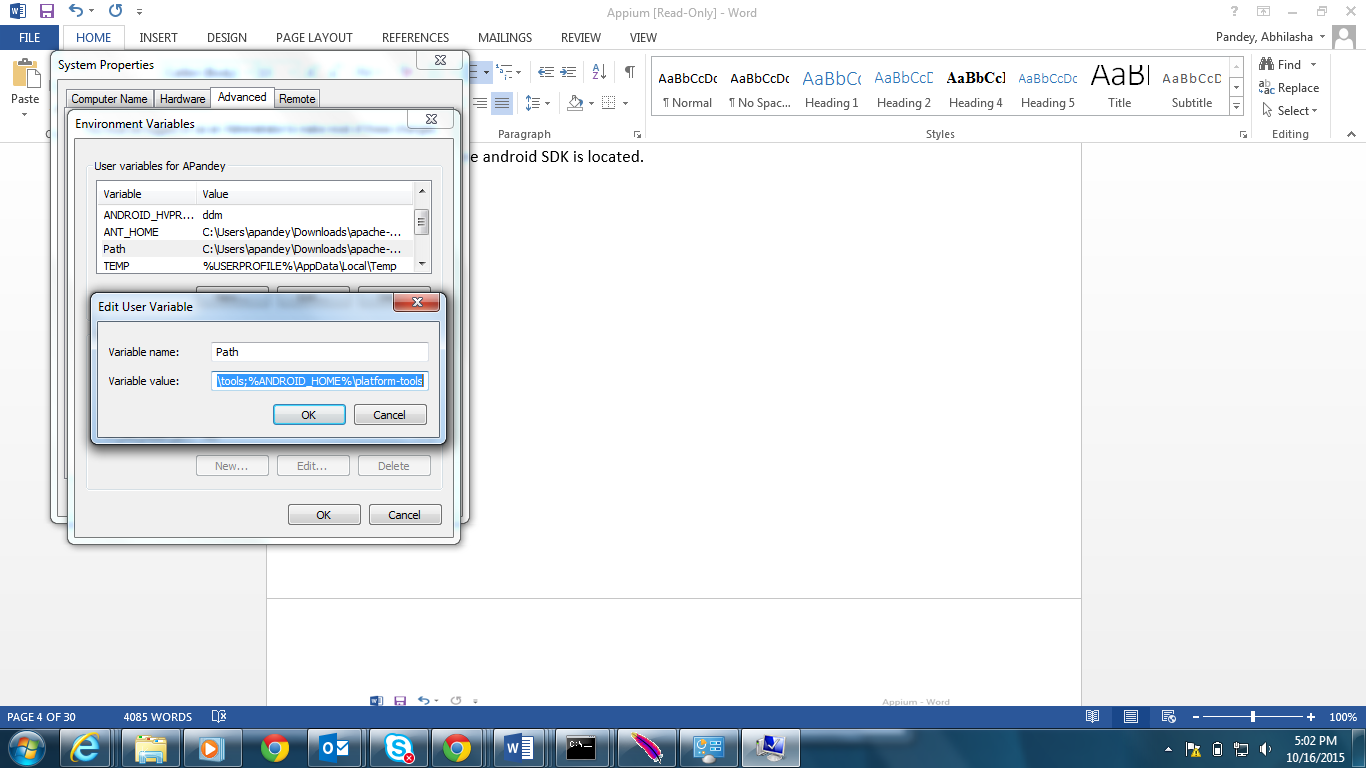


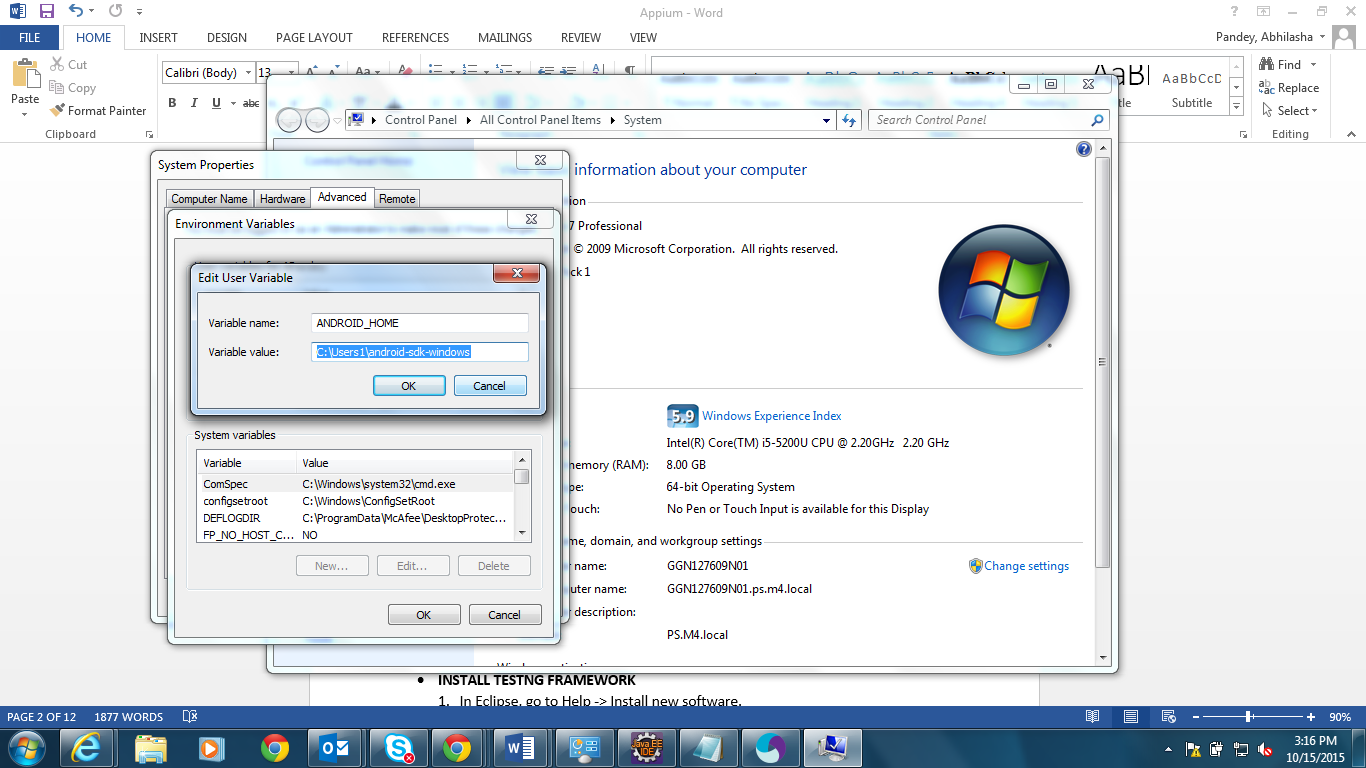
* **Install ADT plugins**

1. Open Eclipse and then go to Help -> Install new software.
2. Enter the following path (<https://dl-ssl.google.com/android/eclipse/> ) in the ‘work with’ field and press ‘add’.
3. Select the options available and click on next.
4. Then go to Windows-> Preferences and select android, give the android SDK path.
5. Download stand-alone SDK tool

(URL: <https://developer.android.com/sdk/installing/index.html>)

1. Once the android SDK is downloaded the environment variable is set by creating a new user variable ANDROID\_HOME and setting its value as path where android SDK is located.





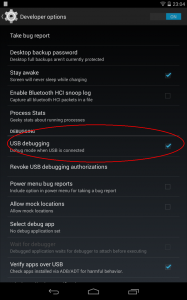
1. Edit the path variable and add the path of tools and platform tools at the end of the path.

For ex-C:\Users\apandey\Downloads\apache-ant-1.9.6-bin\apache-ant-1.9.6\bin;%ANDROID\_HOME%\tools;%ANDROID\_HOME%\platform-tools)

1. Open command prompt and type ‘android’. A window will open. Select all the packages you want to install and click on install packages.
2. Once installation is finished restart Eclipse.

* **DEVELOPER OPTIONS**

1. Go to Settings -> about phone.
2. Click on the build version seven times to enable developer options
3. Switch on developer options and enable USB debugging and stay awake options.



* **INSTALL NODE.JS AND MICROSOFT .NET FRAMEWORK**
* **INSTALL TESTNG FRAMEWORK**

1. In Eclipse, go to Help -> Install new software.
2. Type ‘install TestNG on google, select the first link and pick the path to be used depending on the eclipse version being used.

(URL: <http://testng.org/doc/eclipse.html>).

1. Import TestNG libraries.

* **DOWNLOAD APPIUM**

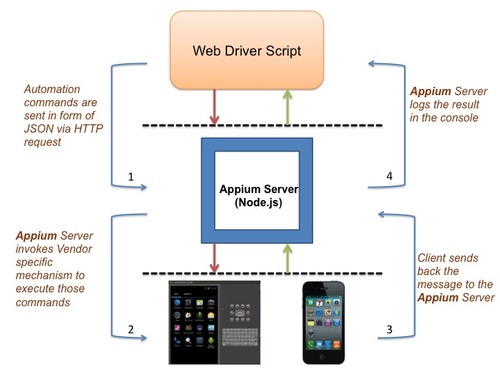


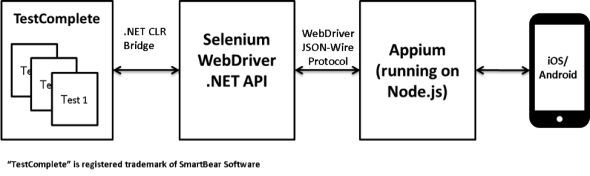
* **IMPORTING JARS**

1. Import jar for selenium webdriver and appium(java-client.jar).

**APPIUM DESIGN CONCEPT**

* Appium is an 'HTTP Server' written using Node.js platform and drives iOS and Android session using Webdriver JSON wire protocol. Hence, before initializing the Appium Server, Node.js must be pre-installed on the system.
* When Appium is downloaded and installed, then a server is setup on our machine that exposes a REST API.
* It receives connection and command request from the client and execute that command on mobile devices (Android / iOS).
* It responds back with HTTP responses. Again, to execute this request, it uses the mobile test automation frameworks to drive the user interface of the apps. Framework like:-
  + Apple Instruments for iOS (Instruments are available only in Xcode 3.0 or later with OS X v10.5 and later)
  + Google UIAutomator for Android API level 16 or higher
  + Selendroid for Android API level 15 or less





**WORKING WITH APPIUM**

BASIC PROGRAM IN APPIUM

**import** io.appium.java\_client.AppiumDriver;

**import** io.appium.java\_client.android.AndroidDriver;

**import** io.appium.java\_client.remote.MobileCapabilityType;

**import** java.net.MalformedURLException;

**import** java.net.URL;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.remote.DesiredCapabilities;

**import** org.testng.annotations.\*;

**public** **class** CalculatorTest {

AppiumDriver driver;

@BeforeClass

**public** **void** setUp() **throws** MalformedURLException{

//Set up desired capabilities and pass the Android app-activity and app-package to Appium

DesiredCapabilities capabilities = **new** DesiredCapabilities();

capabilities.setCapability(MobileCapabilityType.***VERSION***, "4.4.4");

capabilities.setCapability(MobileCapabilityType.***PLATFORM\_NAME***, "Android");

capabilities.setCapability(MobileCapabilityType.***DEVICE\_NAME***,"TA64301YDK");

capabilities.setCapability(MobileCapabilityType.***APP\_PACKAGE***, "com.android.calculator2"); // This is package name of your app (you can get it from apk info app

capabilities.setCapability(MobileCapabilityType.***APP\_ACTIVITY***, "com.android.calculator2.Calculator"); // This is Launcher activity of your app (you can get it from apk info app)

//Create AndroidDriver instance and connect to the Appium server.

//It will launch the Calculator App in Android Device using the configurations specified in Desired Capabilities

driver = **new** AndroidDriver(**new** URL("http://127.0.0.1:4723/wd/hub"), capabilities);

}

@Test

**public** **void** testCal(){

//locate the Text on the calculator by using By.name()

WebElement two=driver.findElement(By.*name*("2"));

two.click();

WebElement plus=driver.findElement(By.*name*("+"));

plus.click();

WebElement four=driver.findElement(By.*name*("4"));

four.click();

WebElement equalTo=driver.findElement(By.*name*("="));

equalTo.click();

//locate the edit box of the calculator by using By.className()

WebElement results=driver.findElement(By.*className*("android.widget.EditText"));

//Check the calculated value on the edit box

**assert** results.getText().equals("6"):"Actual value is : "+results.getText()+" did not match with expected value: 6";

}

@AfterClass

**public** **void** teardown(){

//close the app

driver.closeApp();

}

}

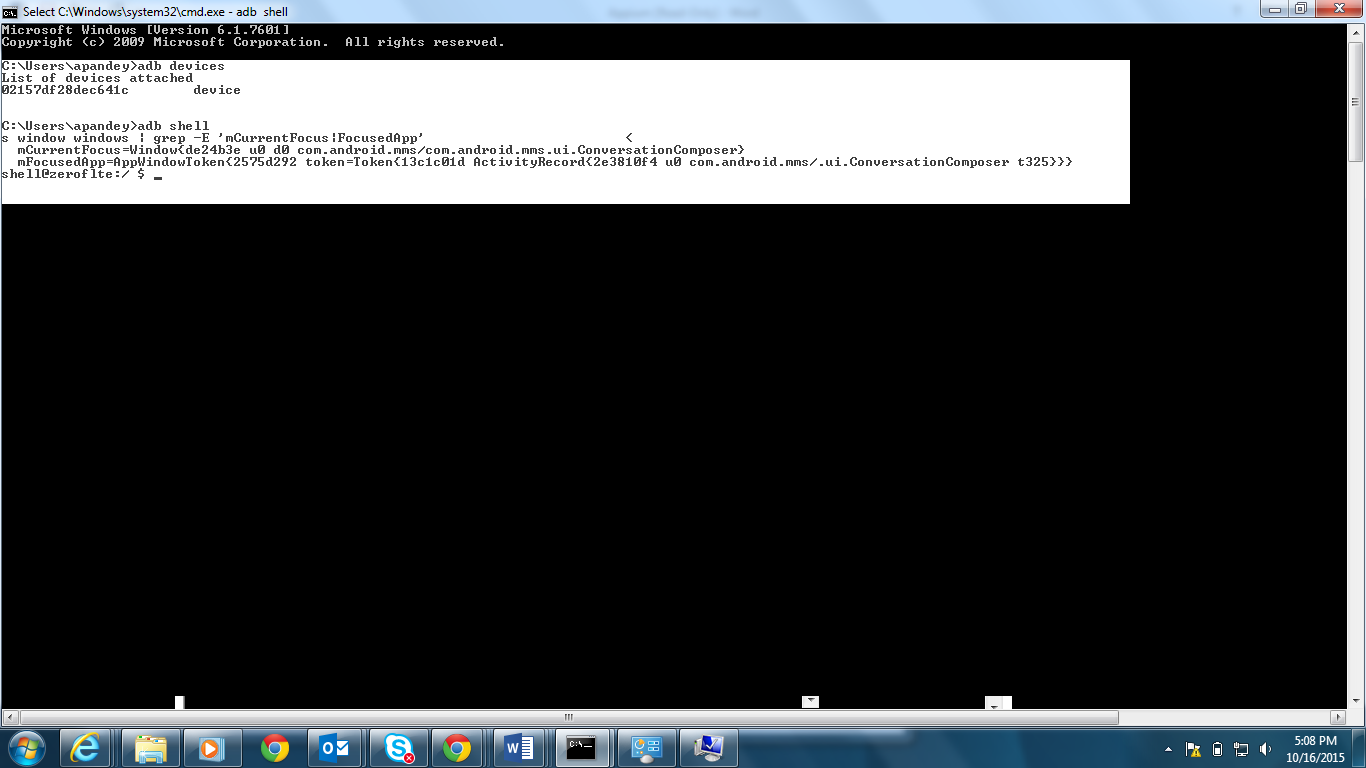
Desired capabilities are a set of keys and values (i.e., a map or hash) sent to the Appium server to tell the server what kind of automation session we’re interested in starting up. There are also various capabilities which can modify the behavior of the server during automation.

Version mentions the android SDK version of the device being used. Platform mentions the platform of the device, that is, android or iOS. App package mentions the package name of the app to be automated and app activity is the name of the activity that you want to launch from the app package. Both app package and app activity are android only capabilities.

The app package and app activity name can be found out using the following steps-

* Launch the app in the simulator or the real device for which the package name and activity name is required.
* Open command prompt and type ‘adb devices’ and press enter to check whether the device is connected or not.
* Then type ‘adb shell’ and press enter followed by -

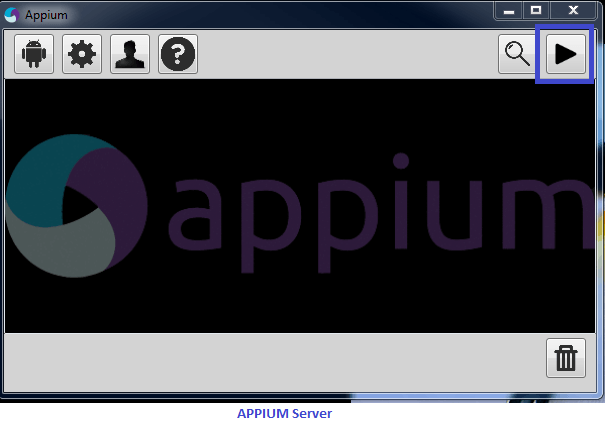
dumpsys window windows | grep -E 'mCurrentFocus|FocusedApp'



After this an android driver is created where the server address and the desired capabilities are passed as parameters.

LAUNCHING APPIUM

To launch Appium, open Appium and then go to Appium Setting and select no reset option. Then click on play button to start appium server. The program should run only after the Appium server is on.



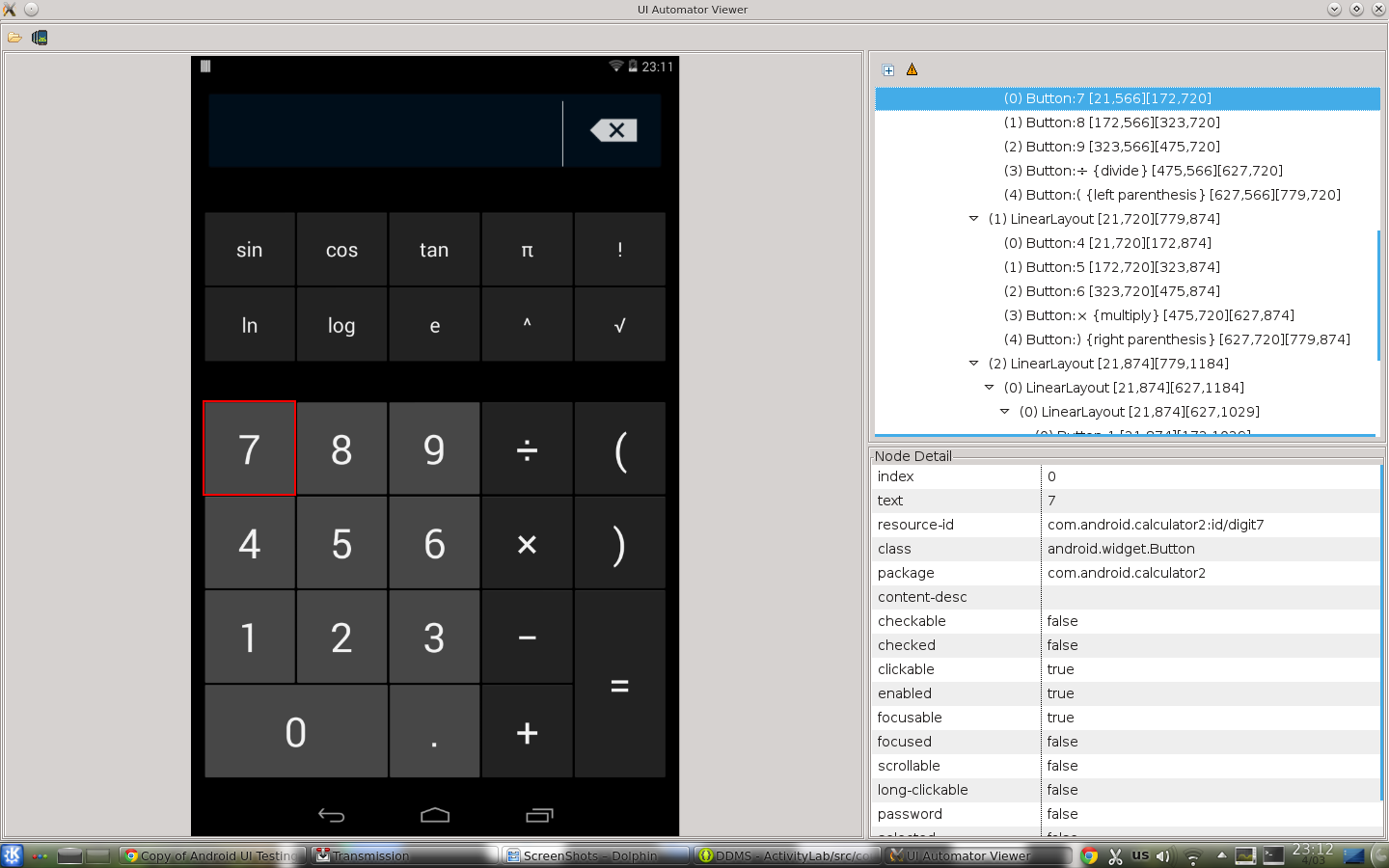
IDENTIFYING ELEMENTS IN APPIUM

In Appium elements can be identified using uiautomatorviever. It GUI tool to scan and analyse the UI components of an Android application.

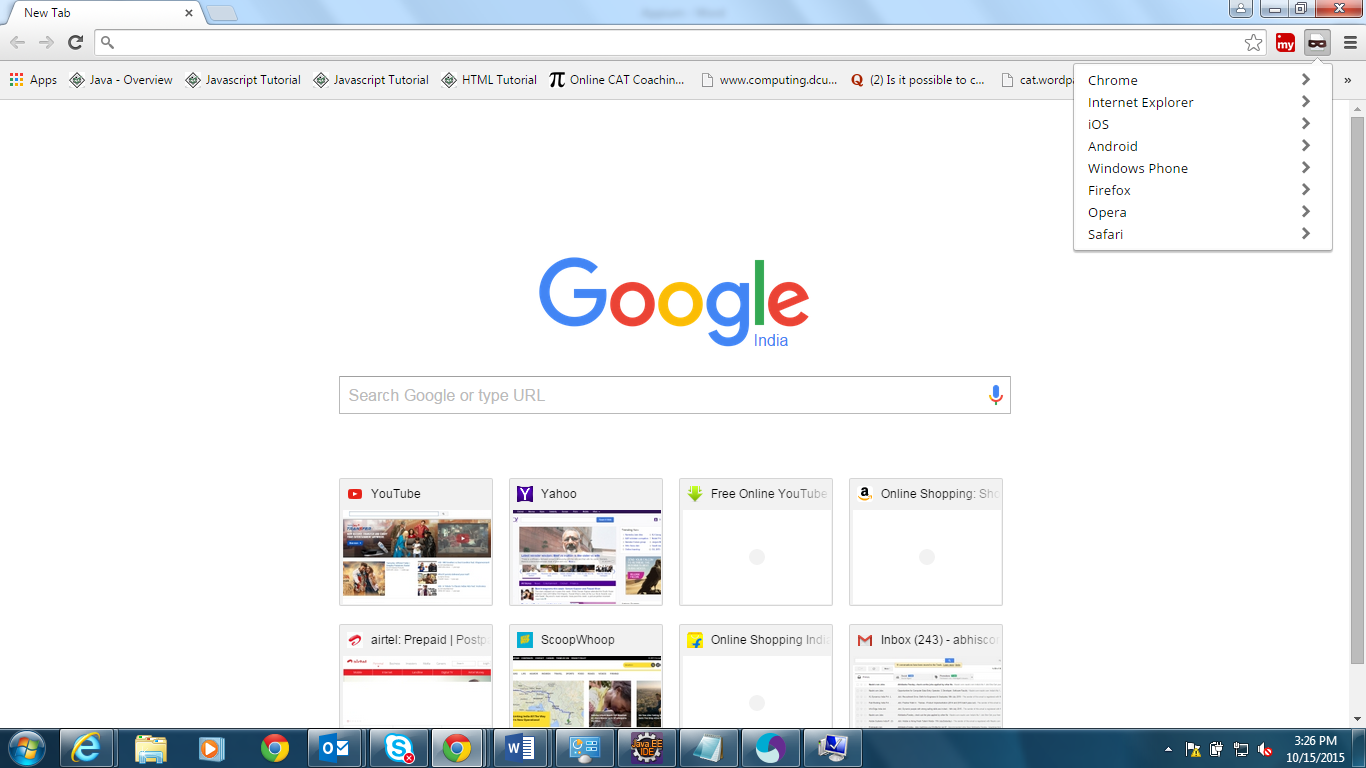
The uiautomatorviewer tool provides a convenient visual interface to inspect the layout hierarchy and view the properties of the individual UI components that are displayed on the test device. Using this information, you can later create uiautomator tests with selector objects that target specific UI components to test.

Steps to launch uiautomatorviewer are-

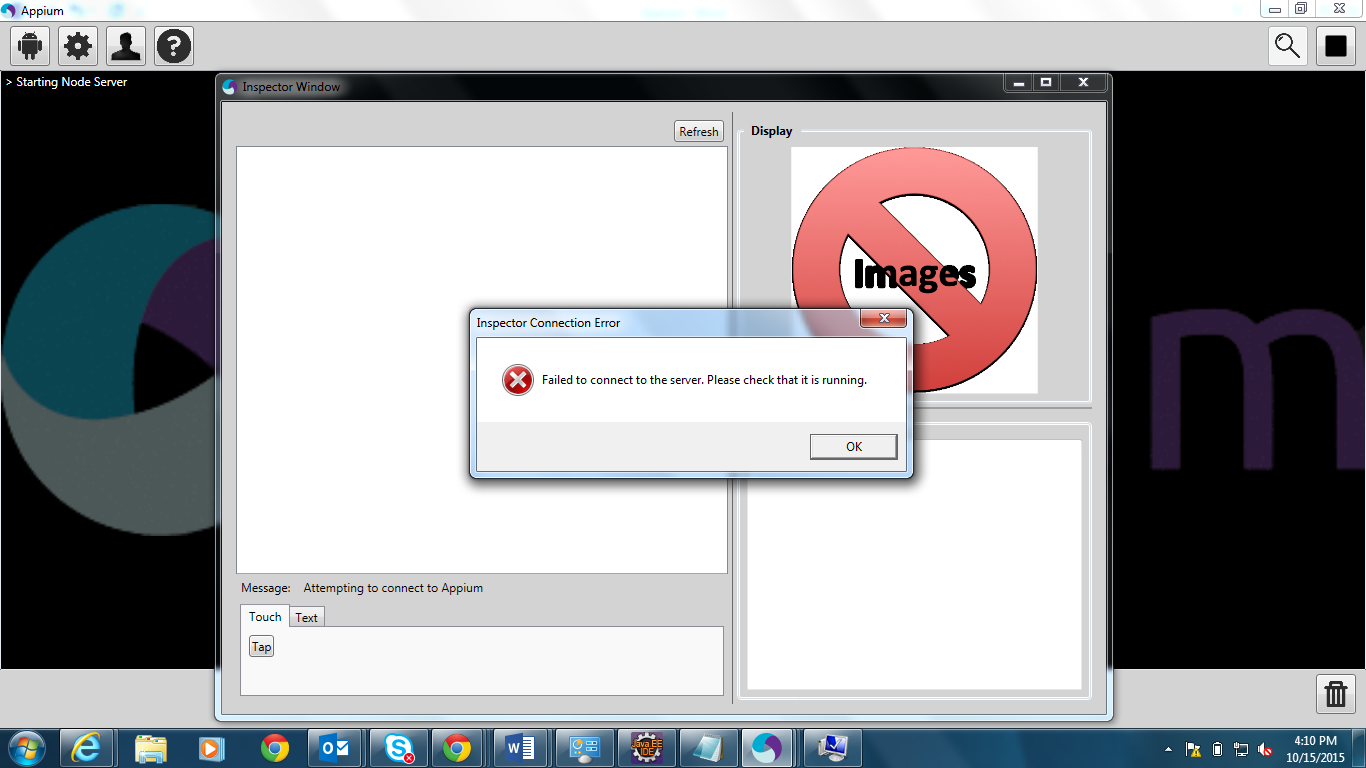
* Open command prompt and type uiautomatorviewer.
* A window opens, click on the device icon located on the top right corner.
* Hover the mouse over the XML screenshot to identify the elements through various properties like id, class, package, text, content-desc etc.



Uiautomatorviewer can be used only for native, hybrid and web based applications. In case of web application if an element cannot be identified through uiautomatorviewer, in that case UserAgent plugin/add-on available in Mozzila Firefox and Google Chrome can be used to identify element using xpath. UserAgent Switcher extension adds a menu and a toolbar button to switch the user agent of a browser.



Another method of identifying element in Appium is using Appium Inspector. It is launched through the appium window. But in Windows operating system Appium Inspector cannot be launched and throws an error as “Failed to connect to server. Check if it is running” even when the appium server is running.



**VARIOUS IMPLEMENTATIONS USING APPIUM**

**AppiumDriver class**

1. Swipe :

Allows to swipe from one location to another. The format for the function is-

Driver.swipe(int startx,int starty,int endx,int endy,duration)

Where,

Startx: starting x coordinate

Starty: starting y coordinate

Endx: ending x coordinate

Endy: ending y coordinate

Duration: the duration for which the swipe function will work

1. Tap :

It is use to tap on a particular element. Format for tap function is :

Driver.tap(int fingers, int x, int y, int duration)

Where,

Fingers: num of fingers used

X: x coordinate location

Y: y coordinate location

Duration: for how long the tap function will work

All the other functions under AppiumDriver class like openApp, closeApp, hideKeyboard etc work fine. The function pinch and zoom do not work.

**TouchActions Class**

Implements actions for touch enabled devices, reusing the available composite and builder design patterns from Actions.

1. Press:-

Allows the user to press at a given location. The location is passed as parameters. The format for press function is –

Driver.press(int x, int y);

Where,

X: x coordinate

Y: y coordinate

1. moveTo:-

Moves the cursor from one location to the specified location. The format for the moveTo function is :

Driver. moveTo(int x, int y)

Where,

X: x coordinate where the cursor needs to be moved

Y: y coordinate where the cursor needs to be moved

1. longPress:

Press and hold the at an absolute position on the screen until the context menu event has fired. The format for the longPress function is:

Driver.longPress(int x, int y, int duration);

Where,

X: x coordinate where the long press needs to be performed

Y: y coordinates where the long press needs to be performed

Duration: for how long the long press is to be executed

1. scrollTo :

ScrollTo function allows the user to scroll to the string which is passed in the function. The format for scrollTo function is:

Driver.scrollTo(String)

Where,

String: the string till which the cursor needs to scrolled.

**MultiTouchActions class**

MultiTouchAction object is a collection of TouchAction objects (remember that TouchAction objects are in turn, a chain of individual actions) Add multiple TouchAction objects using the add() method. When perform() method is called, all actions are sent to the driver. The driver performs the first step of each TouchAction object simultaneously as a multi-touch "execution group". Conceptually, the number of TouchAction objects added to the MultiTouchAction is equal to the number of "fingers" or other appendages or tools touching the screen at the same time as part of this multi-gesture. Then the driver performs the second step of each TouchAction object and another "execution group", and the third, and so on. Using a waitAction() action within a TouchAction takes up one of the slots in an "execution group", so these can be used to sync up complex actions. Calling perform() sends the action command to the Mobile Driver. Otherwise, more and more actions can be chained.

1. Add:

Add a TouchAction to this multi-touch gesture.The format for the add function implementation is:

mAction.add([TouchAction](http://appium.github.io/java-client/io/appium/java_client/TouchAction.html" \o "class in io.appium.java_client) action)

1. Perform :

To perform the multiple actions which have been added.

**AndroidKeyCode interface**

It contains some common key codes for Android Key Events. It is used to carry out various key events such as keyboard keys, home key, app switch etc.

**VARIOUS FLOWS AUTOMATED USING APPIUM**

* To perform addition on a calculator

**import** io.appium.java\_client.android.AndroidDriver;

**import** java.io.File;

**import** java.net.MalformedURLException;

**import** java.net.URL;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

//import org.openqa.selenium.remote.CapabilityType;

**import** org.openqa.selenium.remote.DesiredCapabilities;

//import org.openqa.selenium.remote.RemoteWebDriver;

**import** org.testng.annotations.\*;

**public** **class** Calc1 {

WebDriver driver;

@BeforeClass

**public** **void** setUp() **throws** MalformedURLException, InterruptedException{

//Set up desired capabilities and pass the Android app-activity and app-package to Appium

DesiredCapabilities capabilities = **new** DesiredCapabilities();

capabilities.setCapability("BROWSER\_NAME", "Android");

//capabilities.setCapability("VERSION", "4.2.2");

capabilities.setCapability("deviceName","Android");

capabilities.setCapability("platformName","Android");

File file = **new** File("C://Users//apandey//Downloads//Calculator.apk");

capabilities.setCapability("app",file.getAbsolutePath());

capabilities.setCapability("appPackage", "com.sec.android.app.popupcalculator");

// This package name of your app (you can get it from apk info app)

capabilities.setCapability("appActivity","com.sec.android.app.popupcalculator.Calculator");

Thread.*sleep*(5000);

// This is Launcher activity of your app (you can get it from apk info app)

//Create RemoteWebDriver instance and connect to the Appium server

//It will launch the Calculator App in Android Device using the configurations specified in Desired Capabilities

driver = **new** AndroidDriver(**new** URL("http://127.0.0.1:4723/wd/hub"), capabilities);

}

@Test

**public** **void** testCal() **throws** Exception {

//locate the Text on the calculator by using By.name()

WebElement two=driver.findElement(By.*id*("com.sec.android.app.popupcalculator:id/bt\_02"));

//UiObject button = new UiObject(new UiSelector().text("2"));

two.click();

WebElement plus=driver.findElement(By.*name*("+"));

plus.click();

WebElement four=driver.findElement(By.*name*("4"));

four.click();

WebElement equalTo=driver.findElement(By.*name*("="));

equalTo.click();

//locate the edit box of the calculator by using By.tagName()

WebElement results=driver.findElement(By.*className*("android.widget.EditText"));

//Check the calculated value on the edit box

**assert** results.getText().equals("10"):"Actual value is : "+results.getText()+" did not match with expected value: 6";

}

@AfterClass

**public** **void** teardown(){

//close the app

driver.quit();

}

}

The above program performs addition on calculator application which is installed on the device. In case using a simulator, the app can be installed on the simulator using command prompt. The steps to be followed are:

1. Open command prompt and using cd command reach the folder where the apk of the app is located.
2. Then type ‘adb install’ followed by the name of the apk file. Execute the command to install the app.

In case using a real device the path of the apk file is not required if the app is already installed on the device. The no reset option should be selected in the appium setting so that the app is not reinstalled if already present.

It is very important to mention correct package and activity name else the app will not be launched. There should be no spaces between the name of package/activity and the semi-colon else an exception known as session not created is thrown (org.openqa.selenium.SessionNotCreatedException).

It is should be made sure that the appium server is started before executing the program.

In the above program the elements are identified using uiautomatorviewer. The text property being unique is used for identifying the element.

* To automate a hybrid app (ChangePoint Timesheet)

**import** io.appium.java\_client.android.AndroidDriver;

**import** io.appium.java\_client.remote.MobileBrowserType;

**import** java.io.File;

**import** java.net.MalformedURLException;

**import** java.net.URL;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

//import org.openqa.selenium.remote.CapabilityType;

**import** org.openqa.selenium.remote.DesiredCapabilities;

//import org.openqa.selenium.remote.RemoteWebDriver;

**import** org.testng.annotations.\*;

**public** **class** New\_Example {

AndroidDriver driver;

@BeforeClass

**public** **void** setUp() **throws** MalformedURLException{

//Set up desired capabilities and pass the Android app-activity and app-package to Appium

DesiredCapabilities capabilities = **new** DesiredCapabilities();

capabilities.setCapability("BROWSER\_NAME", "Android");

//capabilities.setCapability("VERSION", "5.1.1");

capabilities.setCapability("deviceName","Nexus 5");

capabilities.setCapability("platformName","Android");

capabilities.setCapability("browserName", "chrome");

driver = **new** AndroidDriver(**new** URL("http://127.0.0.1:4723/wd/hub"), capabilities);

}

@Test

**public** **void** testCal() **throws** Exception {

WebElement x = **null**;

WebElement y= **null**;

driver.get("https://lochbridge.changepointasp.com/");

Thread.*sleep*(10000);

//locate the Text on the calculator by using By.name()

//WebElement username=driver. findElement(By.name("someone@example.com"));

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

//UiObject button = new UiObject(new UiSelector().text("2"));

username.sendKeys("abhilasha.pandey@lochbridge.com");

WebElement password =driver.findElement(By.*id*("passwordInput"));

password.sendKeys("ap1rp2sp3@!");

WebElement sign\_in=driver.findElement(By.*id*("submissionArea"));

sign\_in.submit();;

Thread.*sleep*(10000);

//to select timesheet

WebElement elements = driver.findElementByAndroidUIAutomator("new UiSelector().description('My time sheet')");

elements.click();

Thread.*sleep*(10000);

//to select the day

x= driver.findElement(By.*xpath*("//\*[@id='day1']"));

x.click();

// to select the actions opt

x= driver.findElement(By.*xpath*("//\*[@id='headTimeSheet']/a[2]"));

x.click();

Thread.*sleep*(5000);

// to select the time entry opt

x=driver.findElement(By.*xpath*("//\*[@id='liAddTime']/a"));

x.click();

Thread.*sleep*(5000);

//to select task type option

x=driver.findElement(By.*xpath*("//\*[@id='Master\_cpMobileBody\_selTimeType']"));

x.click();

// to select the option

// x.sendKeys("open task");

Thread.*sleep*(5000);

x = driver.findElement(By.*id*("Master\_cpMobileBody\_ddParentEntity\_txtSearch"));

x.sendKeys("Training and Development");

Thread.*sleep*(2000);

x = driver.findElement(By.*id*("Master\_cpMobileBody\_ddParentEntity\_ImgSearch"));

x.click();

Thread.*sleep*(2000);

x = driver.findElement(By.*id*("Master\_cpMobileBody\_ddTask\_ImgSearch"));

x.click();

Thread.*sleep*(2000);

x = driver.findElement(By.*id*("Master\_cpMobileBody\_selRegularHours"));

x.sendKeys("8");

Thread.*sleep*(2000);

x = driver.findElement(By.*id*("Master\_ancSave"));

x.click();

Thread.*sleep*(2000);

}

@AfterClass

**public** **void** teardown(){

//close the app

driver.quit();

}

}

This is an example of an app where a switching from hybrid to native is required. In the above example since the properties of the elements are not uniquely identifiable, the elements are found using UserAgent and the xpath generated using them. UserAgent is used only for web elements and cannot be used for native elements.

The example also showcase that it is possible to switch from hybrid to native without editing any properties. This was not possible in the earlier versions of Appium.

* To automate swipe, ScrollTo and MultiTouchActions

**import** io.appium.java\_client.AppiumDriver;

**import** io.appium.java\_client.android.AndroidDriver;

**import** io.appium.java\_client.android.AndroidKeyCode;

**import** io.appium.java\_client.remote.MobileBrowserType;

**import** java.io.File;

**import** java.net.MalformedURLException;

**import** java.net.URL;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.Point;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

//import org.openqa.selenium.remote.CapabilityType;

**import** org.openqa.selenium.remote.DesiredCapabilities;

//import org.openqa.selenium.remote.RemoteWebDriver;

**import** org.testng.annotations.\*;

**import** android.graphics.Point.\*;

**public** **class** ZoomClass2 {

AndroidDriver driver;

WebElement wb=**null**;

**public** **int** generate(Point point)

{

**int** x = point.getY() ;

**return** x;

}

@BeforeClass

**public** **void** setUp() **throws** MalformedURLException, InterruptedException{

//Set up desired capabilities and pass the Android app-activity and app-package to Appium

DesiredCapabilities capabilities = **new** DesiredCapabilities();

capabilities.setCapability("BROWSER\_NAME", "Android");

//capabilities.setCapability("VERSION", "5.1.1");

capabilities.setCapability("deviceName","Nexus 5");

capabilities.setCapability("platformName","Android");

// capabilities.setCapability("browserName", "chrome");

capabilities.setCapability("appPackage", "com.android.settings");

capabilities.setCapability("appActivity", "com.android.settings.Settings");

driver = **new** AndroidDriver(**new** URL("http://127.0.0.1:4723/wd/hub"), capabilities);

Thread.*sleep*(5000);

}

@Test

**public** **void** testCal() **throws** Exception {

wb=driver.scrollTo("Developer options");

Thread.*sleep*(5000);

wb.click();

Thread.*sleep*(5000);

driver.tap(1, wb, 500);

driver.pressKeyCode(187);

Thread.*sleep*(5000);

}

@AfterMethod

**public** **void** tearDown() {

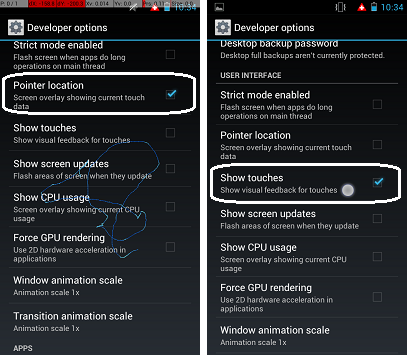
driver.quit();

}

}

In this example, the scrollTo functionality is implemented. In this program, it was learnt that it is not possible to open any sub setting for any phone using capabilities and the app package and activity name. Therefore, in this case scrollTo function is used to access the sub settings. The scrollTo function is followed by the tap function which is used for clicking on the identified web element.The program also showcase use of androidKeyCode. The code passed in the function is used for app\_switch.

In many functions like swipe, tap, longPress, moveTo, press, as parameters we need to pass pointer locations. These pointer locations can be identified by enabling ‘pointer location’ option in Developer Options. The option allows to know the pointer location of an element once the mouse is hovered over it. The Dx and Dy located on the top left corner give the x and y pointer location respectively.



The program implementing moveTo,press and multiTouch Action is as follows-

**import** io.appium.java\_client.AppiumDriver;

**import** io.appium.java\_client.MultiTouchAction;

**import** io.appium.java\_client.TouchAction;

**import** io.appium.java\_client.android.AndroidDriver;

**import** io.appium.java\_client.android.AndroidKeyCode;

**import** io.appium.java\_client.remote.MobileBrowserType;

**import** java.io.File;

**import** java.net.MalformedURLException;

**import** java.net.URL;

**import** java.util.List;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.Point;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

//import org.openqa.selenium.remote.CapabilityType;

**import** org.openqa.selenium.remote.DesiredCapabilities;

//import org.openqa.selenium.remote.RemoteWebDriver;

**import** org.testng.annotations.\*;

**import** com.android.uiautomator.core.UiSelector;

**import** android.graphics.Point.\*;

**public** **class** Multi\_Touch {

AndroidDriver driver;

WebElement wb=**null**;

**public** **int** generate(Point point)

{

**int** x = point.getY() ;

**return** x;

}

@BeforeClass

**public** **void** setUp() **throws** MalformedURLException, InterruptedException{

//Set up desired capabilities and pass the Android app-activity and app-package to Appium

DesiredCapabilities capabilities = **new** DesiredCapabilities();

capabilities.setCapability("BROWSER\_NAME", "Android");

//capabilities.setCapability("VERSION", "5.1.1");

capabilities.setCapability("deviceName","Nexus 5");

capabilities.setCapability("platformName","Android");

// capabilities.setCapability("browserName", "chrome");

capabilities.setCapability("appPackage", "com.android.settings");

capabilities.setCapability("appActivity", "com.android.settings.Settings");

driver = **new** AndroidDriver<WebElement>(**new** URL("http://127.0.0.1:4723/wd/hub"), capabilities);

Thread.*sleep*(5000);

}

@Test

**public** **void** testCal() **throws** Exception {

wb=driver.findElement(By.*name*("Display"));

wb.click();

wb=driver.findElement(By.*name*("Brightness level"));

wb.click();

wb= driver.findElement(By.*id*("com.android.systemui:id/slider"));

MultiTouchAction multiTouch = **new** MultiTouchAction(driver);

TouchAction tAction0 = **new** TouchAction(driver);

tAction0.press(412,225).waitAction(1000).moveTo(809,225).release();//press

multiTouch.add(tAction0);

multiTouch.perform();// now perform both the actions

}

@AfterClass

**public** **void** teardown(){

//close the app

driver.quit();

}}

In the above example, sub setting are opened in order to increase brightness of the devices using the slidebar. The pointer locations are found out using the pointer locations option under Developer Options which is in turn used in the moveTo and press function. In both moveTo and press function the x and y pointer location is passed as parameters.

Though multiTouch Actions class is used to perform multiple action simultaneously but it could not be used for the same though it was successful in implementing single actions.

The program implementing swipe function is as follows-

import io.appium.java\_client.MultiTouchAction;

import io.appium.java\_client.TouchAction;

import io.appium.java\_client.android.AndroidDriver;

import java.net.MalformedURLException;

import java.net.URL;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.remote.DesiredCapabilities;

import org.testng.annotations.AfterClass;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.Test;

public class ZoomTest {

public AndroidDriver driver;

  WebElement element;

@BeforeClass

public void setUp() throws MalformedURLException{

       DesiredCapabilities capabilities = new DesiredCapabilities();

       capabilities.setCapability("VERSION", "5.1.1");

       capabilities.setCapability("deviceName","Samsung S6");

       capabilities.setCapability("platformName","Android");

 //    capabilities.setCapability("appPackage" , "com.sec.android.gallery3d");

//               capabilities.setCapability("appActivity" , "com.sec.android.gallery3d.app.GalleryOpaqueActivity");

                   capabilities.setCapability("appPackage" , "com.cyngn.gallerynext");

                   capabilities.setCapability("appActivity" , "com.cyngn.gallerynext.app.GalleryActivity");

       driver = new AndroidDriver(new URL("<http://127.0.0.1:4723/wd/hub>"), capabilities);

}

@Test

public void zoom1() throws InterruptedException

{

                driver.swipe(550, 1850 , 550 , 100 , 15000);

                WebElement element = driver.findElementByAndroidUIAutomator("new UiSelector().index(5)");

                element.click();

                MultiTouchAction multiTouch = new MultiTouchAction(driver);

                TouchAction tAction0 = new TouchAction(driver);

                TouchAction tAction1 = new TouchAction(driver);

                tAction0.longPress(600 , 800).waitAction(1000).moveTo(780,650).release();

                tAction1.longPress(500,1100).waitAction(1000).moveTo(380,1140).release();

                multiTouch.add(tAction0).add(tAction1);

                multiTouch.perform();

                Thread.sleep(10000);

}

@AfterClass

public void Quit(){

       driver.quit();

}

}

The above program opens gallery on the real device and then using swipe function goes to the picture that needs to be selected. The starting x location , starting y location , ending x location, ending y location and the durartion is passed as parameter in the swipe function in the same order.

The program also tries to implement zoom through multiTouch Actions but the same could not be achieved though there is no error in the code.

**CONCLUSION**

* **PROS OF USING APPIUM**

1. Supports both android and iOS

This is one of the biggest advantage of appium is that it supports both android and iOS which covers the maximum market in terms of usage of various operating system for mobile phones. Unfortunately, there is no support for windows phones.

1. Appium is based on Selenium and supports various platforms

Appium is based on Selenium which is an HTTP protocol by Google designed to automate browsers. The idea is actually very nice as automating an app (especially a webview-based one) is not so different (in terms of required APIs) from automating a browser. Appium is also designed to encourage a 2-tier architecture: a machine runs the test written in one language ([csharp], [ruby], [javascript] are only a few among the many supported ones) and another one (the test server) actually executes it. Furthermore the WebDriver protocol targets scalability (because based on [HTTP](http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)), this makes Appium very scalable as well; remember that you will need to **write your test once**, Appium will be in charge of executing it on more platforms.

1. Doesn't require access to your source code or library. One can test what one will actually ship.

Unlike other tools Appium doesn’t require you to include some extra agents in your app to make it automation friendly. It believes in the philosophy of testing the same app which we are going to submit in the app store.

1. Performance

Appium is not a huge application and **requires very little memory**. Its architecture is actually pretty simple and light as Appium acts like a proxy between your test machine and each platform automation toolkit. Once up and running, Appium will listen to HTTP requests from your tests; when a new session is created, a component in Appium's [Node.js](http://nodejs.org/)code called \_proxy\_ will forward these Selenium commands to active platform drivers. In the case of Android for example, Appium will forward incoming commands to the [chromedriver] (90% of cases, Appium will not even change commands while routing them), this happens because ChromeDriver supports WebDriver and Selenium. For this reason Appium will not allocate much memory itself, you will see a lot of memory being allocated by other processes like [adb], ChromeDriver or the iOS automation toolkit (called by Appium while testing and automating).

1. Open source hence free of cost and supports hybrid, native and web applications.
2. Support automation testing both on real device and simulators.

* **CONS OF USING APPIUM**

1. Limited support for Android < 4.1

Appium uses UIAutomator for Android automation which only supports Android SDK Platform, API 16 or higher so to support the older APIs they have used another open source library called [Selendroid](https://github.com/DominikDary/selendroid). So I would not say it as a limitation but it is definitely an overhead on the configuration side.

1. No multiple session handling

Since Appium is a server, it serves HTTP requests; you might have two different computers running a test each against the same Appium server: what happens? As for now, Appium does not support this scenario and the second test will be aborted. This is a considerable limitation, because no queuing system comes with Appium. If one needs to support multiple sessions, they will need to implement this feature by yourself.

1. Exchanging data

Another little nightmare with Appium is **exchanging data**. When your test needs to exchange data with your app (especially in the incoming direction), you will need to play some tricks. So always consider that sending and receiving information is not that straightforward.

1. Limited support for gestures

The gestures such as tap, swipe and scroll can be easily implemented using Appium but gesture such as zoom, pinch and multiTouch cannot be implemented.

1. Appium Inspector not working

Appium’s own inspector for identification of elements does not work for windows. Due to this it can be difficult to identify elements for which the properties are not uniquely identifiable. Since appium inspector does not work, xpath cannot be found for elements of native and hybrid app.

1. Appium set-up

Setting up of appium can be a difficult task since the setup is not that straight-forward.

1. Lack of proper documentation

The documentation provided on Appium’s website is quite weak and a little difficult to follow.

1. Less availability of online tutorials and help.
2. Still in developing phase and therefore can be difficult to use. A lot of unexpected error arise which can be difficult to fix.

**REFERENCES**

* <http://www.guru99.com/introduction-to-appium.html>
* <http://appium.io/documentation.html?lang=en>
* <http://blogs.technet.com/b/antino/archive/2014/11/30/q-amp-a-on-appium-should-i-use-it.aspx#q7>
* <http://stackoverflow.com/questions/28363221/advantages-and-disadvantages-of-appium>
* <http://www.fredberinger.com/2013/12/4-reasons-why-appium-is-not-ready-for-prime-time/>