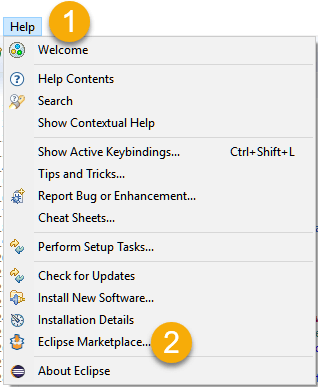
**Installation:**

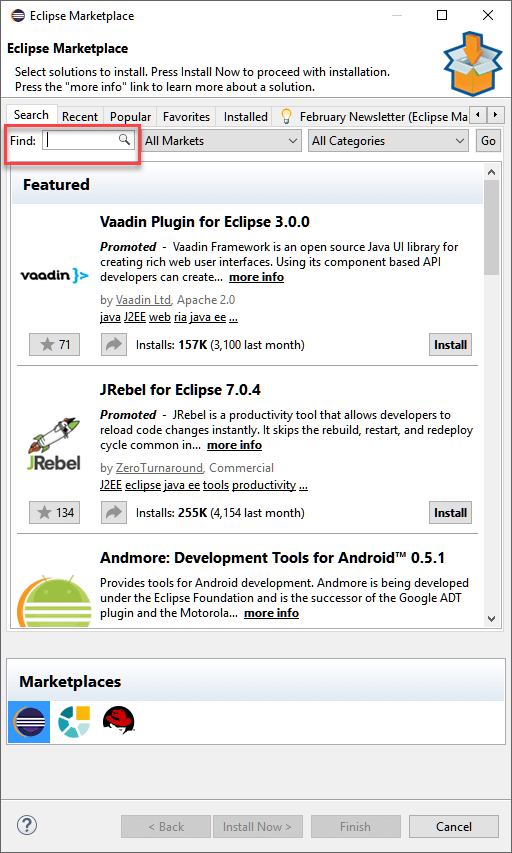
**Part 1: Installing TestNG in Eclipse**

**Step 1:**

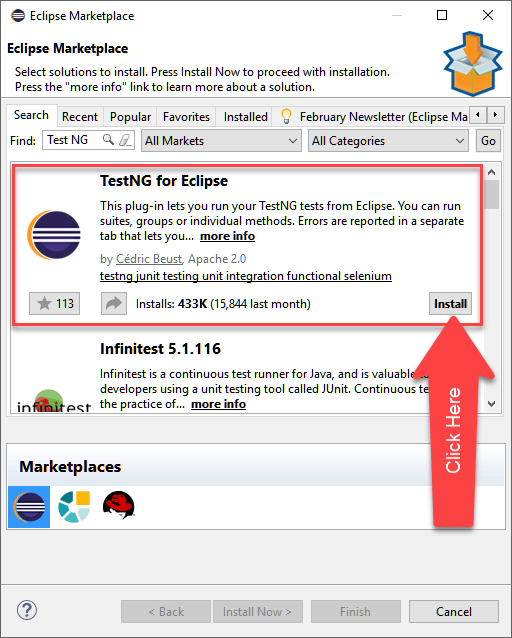
* Launch Eclipse.
* On the menu bar, click Help.
* Choose the "Eclipse Marketplace..." option.



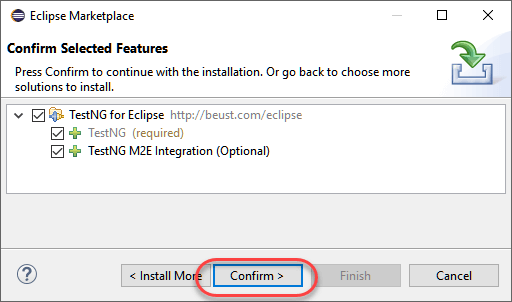
**Step 2:**In the Eclipse Marketplace dialog box, type TestNG in the search box and press the search button( magnifying glass) or press enter key



**Step 3:** Click Install

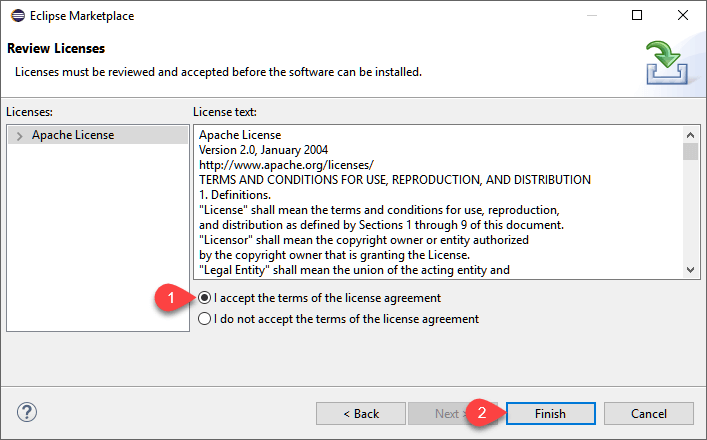


**Step 4:**A new window for feature selection will open, Do not change anything andClick on confirm button

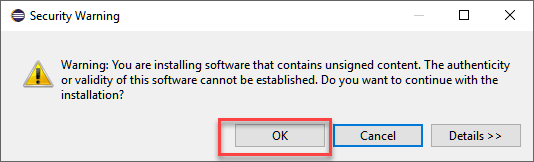


**Step 5:**

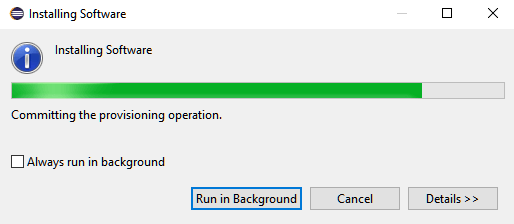
* Click Next again on the succeeding dialog box until you reach the License Agreement dialog.
* Click "I accept the terms of the license agreement" then click Finish.



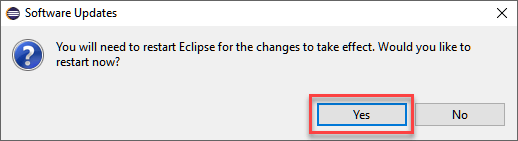
**Step 6:**If you encounter a Security warning, just click OK



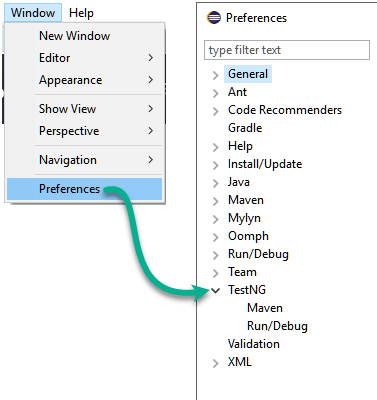
Wait for the installation to finish



**Step 7:**When Eclipse prompts you for a restart, just click Yes.



**Step 8:**After the restart, verify if TestNG was indeed successfully installed. Click Window > Preferences and see if TestNG is included on the Preferences list.



That's it to TestNG Installation

**Part 2: Getting your environment setup with Appium and TestNG**

Surprisingly, the hardest part of mobile automated testing is figuring out which tools to use, which versions to use and how to install everything so that it works seamlessly together. Sadly, it is no where near as easy as performing a few clicks to install everything you need.

This tutorial describes our recommended tools that we use for automated testing of Android applications at [2359 Media](http://www.2359media.com/) — but there are plenty of other variations that might work just as well. Here is a comprehensive list of everything you need on a fresh installation of **OSX**.

* Homebrew
* Carthage
* Node & NPM
* JDK
* Android Studio and Android SDKs
* Appium
* Appium Doctor
* Gradle
* Maven
* Eclipse
* TestNG

**Install Homebrew**

Homebrew is a package management software that will make it much simpler for us to install a few other software.

To install, follow the instructions on this page: <https://brew.sh/>  
This step will also install the Xcode Command Line Tools as part of the process.

**Install Carthage**

Carthage is a dependency manager. In our case, it is required by WebDriverAgent.

In [terminal](https://www.macworld.co.uk/how-to/mac-software/how-use-terminal-on-mac-3608274/), enter the following:

brew install carthage

**Install Node & NPM**

Node is a javascript run-time environment and npm is the node package manager. We need these because Appium is a node application.

In terminal, enter the following: (this command will install npm as well)

brew install node

**Install JDK and set JAVA\_HOME**

As we’ll be writing our tests in Java, we need the Java Development Kit (JDK)

Download the JDK, jdk-8u181-macosx-x64.dmg, from this link: <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>.

Then double click the JDK and follow through all the installation steps.

Next, you’ll need to set JAVA\_HOME in your [.bash\_profile](https://top.quora.com/What-is-bash_profile-and-what-is-its-use). To do so, first find the location of where the JDK was installed by entering the following into terminal:

/usr/libexec/java\_home --v

The terminal output should be along the lines of

/Library/Java/JavaVirtualMachines/jdk1.8.0\_171.jdk/Contents/Home

Next edit your bash\_profile by editing it in vim editor. Type the following into terminal:

vim ~/.bash\_profile

Press the ‘i’ key to enter insert mode then move the cursor a new line and add the following 2 lines

export JAVA\_HOME=[path of your java home]

export PATH=$JAVA\_HOME/bin:$PATH





for example

Press the ‘esc’ key and then type ‘:wq’ and press enter to save and quit vim. Restart Terminal to pick up the new .bash\_profile settings.

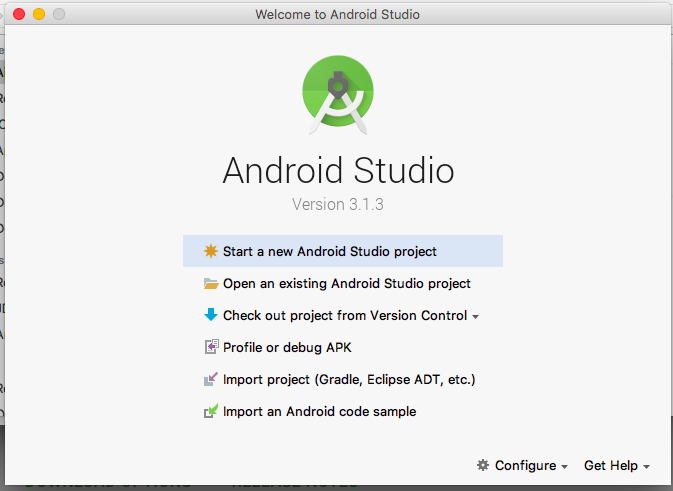
**Install Android Studio & Android SDKs**

We’ll need the Android SDKs next to write and run tests on Android apps and devices. The quickest way to get these installed is via Android Studio.

Download and install Android Studio from <https://developer.android.com/studio/>

Launch Android Studio and complete the setup process until you reach the welcome page.





From the bottom, select “Configure” > “SDK Manager” > “Android SDK”.

For the tab “SDK Platforms”, select the Android SDK Platform versions that you intent to run tests against.

For the tab “SDK Tools” and ensure the following are selected

* Android SDK Build-tools (highest version)
* Android SDK Platform-Tools
* Android SDK Tools
* Intel x86 Emulator Accelerator (HAXM installer)
* Android Support Respository
* Google Repository

Click “OK” to install any of the additions that you selected.

Now copy the value of the Android SDK Location at the top of the SDK Manager screen. This is the value that we’ll need to set for ANDROID\_HOME in our .bash\_profile.

Once again, Type the following into terminal:

vim ~/.bash\_profile

Press the ‘i’ key to enter insert mode then move the cursor a new line and add the following 2 lines

export ANDROID\_HOME=[your Android SDK Location]

export PATH=$PATH:$ANDROID\_HOME/tools:$ANDROID\_HOME/platform-tools





for example

Press the ‘esc’ key and then type ‘:wq’ and press enter to save and quit vim.

Restart terminal to pickup the new .bash\_profile then run the following command to check that it does not throw an error:

adb

**Install Appium**

Appium is an open source test automation framework for use with native, hybrid and mobile web apps. It drives iOS, Android and Windows mobile apps using the WebDriver protocol.

In terminal, enter the following:

npm install -g appium@1.7.2

**Install Appium Doctor**

Appium doctor is a mini software that checks all (well, almost all) of the preconditions for appium to run successfully.

In terminal, enter the following:

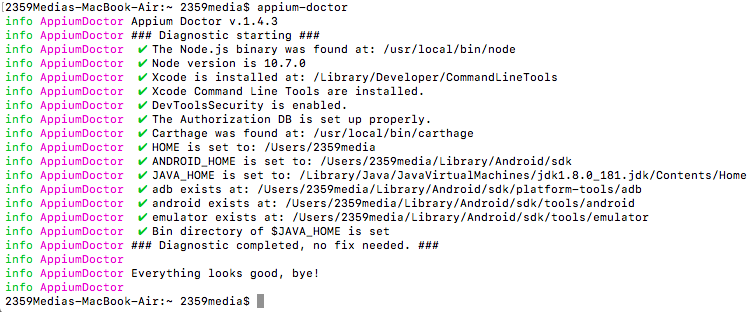
npm install -g appium-doctor

At this point you can run the appium doctor to get a sense of what else we’ll need to get straight before we really get started. To run it, enter the following into terminal:

appium-doctor

appium-doctor’s output should look something like this:





**Install Gradle**

Gradle is an open-source build automation system used to create apk and jar files.

In terminal, enter the following:

brew install gradle

Check the gradle version by entering this into the terminal:

gradle -v

Next edit your .bash\_profile by adding this line to it (follow the steps using vim above)

*export PATH=/usr/local/Cellar/gradle/****4.8.1****/bin:$PATH*

(Be sure to set the version number as per the version you installed)

**Install Maven**

Maven is a tool used for building and managing Java-based projects.

In terminal, enter the following:

brew install maven

Check the maven version by entering this into the terminal:

mvn -version

Next edit your .bash\_profile by adding this line to it (follow the steps using vim above)

*export PATH=/usr/local/Cellar/maven/****3.5.4****/bin:$PATH*

(Be sure to set the version number as per the version you installed)

**Install Eclipse**

Eclipse is the Integrated Development Environment (IDE) that we are going to use to write, compile and run our test scripts.

Download and install **Eclipse IDE for Java EE Developers** from <https://www.eclipse.org/downloads/packages/>

**Install TestNG**

TestNG is a testing framework that we’ll be using to manage our automated test suite.

Launch Eclipse, select “Help” > “Install New Software” from the menu bar.

Paste “<http://beust.com/eclipse>” into the “Work with:” field and press Enter. Eclipse will take a while to fetch the relevant package to display, select the checkbox next to TestNG and click “Next” to complete the dialogs. Eclipse will take a while to install the TestNG package and then prompt you to restart Eclipse.

Alright! Let’s….get started?

**Part 3: Writing your test cases**

If you’ve completed all the steps in Part 1 of this tutorial, your environment should be ready to create and run some automated test scripts. This section will guide you through the following steps:

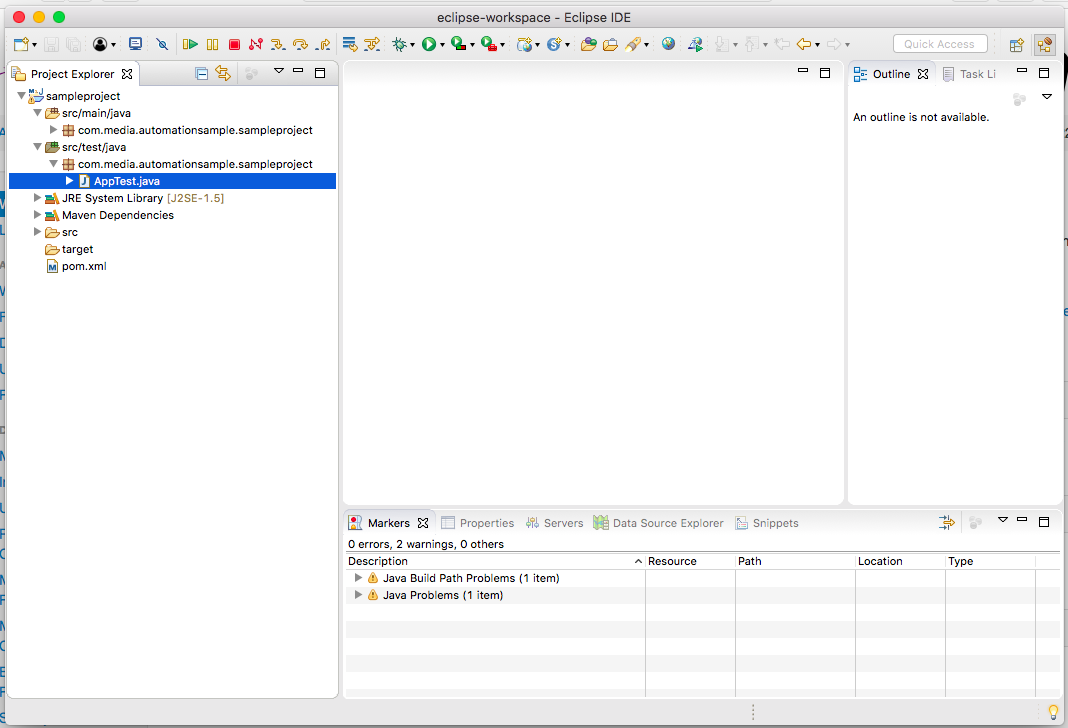
* Setting up your project
* Writing a basic setup script
* Running your basic tests on a real device
* Writing actual tests!

**Setup Your Project**

Launch Eclipse, select “File” > “New” > “Maven Project” and follow through the dialogs with the default settings.

You will be required to enter a Group Id and an Artifact Id. The Group ID will uniquely identity your project across all projects. The Artifact ID is the name of the jar (without version). We recommend reverse-domain-packages for the Group ID (e.g. com.2359media.automation) and the project name for the Artifact ID (e.g. sampleproject). The basic maven project should look like this.





Next, let’s add the required JARs to the project. Double click on the pom.xml file and edit the <dependencies>…</dependencies> block to look like this:

<dependencies>

<dependency>

<groupId>io.appium</groupId>

<artifactId>java-client</artifactId>

<version>5.0.4</version>

</dependency>

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>3.9.1</version>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

<version>3.7</version>

</dependency>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.14.3</version>

<scope>test</scope>

</dependency>

</dependencies>

This defines the dependancies for maven to download into this project. Right click the project, select “Maven > Update Project…” to trigger downloading the dependencies. Once this is done, you should see a lot more .jar files in the Maven Dependencies dropdown of the Eclipse Project Explorer.

**Write A Basic Setup Script**

In the Eclipse file explorer, expand “src/test/java” and expand the package. Double click the file “AppTest.java” and *delete all its contents except the first line.* Paste the following code into the file under the first line.

**import** java.net.MalformedURLException;

**import** java.net.URL;

**import** java.util.concurrent.TimeUnit;

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

**import** org.testng.annotations.BeforeSuite;

**import** org.testng.annotations.AfterSuite;

**import** org.testng.annotations.BeforeTest;

**import** org.testng.annotations.Test;

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

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

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

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

**public** **class** AppTest {

**public** **static** URL *url*;

**public** **static** DesiredCapabilities *capabilities*;

**public** **static** AndroidDriver<MobileElement> *driver*;

//1

@BeforeSuite

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

//2

**final** String URL\_STRING = "<http://127.0.0.1:4723/wd/hub>";

*url* = **new** URL(URL\_STRING);

*//3*

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

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

*capabilities*.setCapability(MobileCapabilityType.***APP***, "<https://github.com/afollestad/material-dialogs/raw/master/sample/sample.apk>");

*capabilities*.setCapability(MobileCapabilityType.***NO\_RESET***, **true**);

*capabilities*.setCapability(MobileCapabilityType.***AUTOMATION\_NAME***, "UiAutomator2");

*//4*

*driver* = **new** AndroidDriver<MobileElement>(*url*, *capabilities*);

*driver*.manage().timeouts().implicitlyWait(2, TimeUnit.***SECONDS***);

*driver*.resetApp();

}

//5

@AfterSuite

**public** **void** uninstallApp() **throws** InterruptedException {

*driver*.removeApp("com.example.android.contactmanager");

}

//6

@Test (enabled=**true**) **public** **void** myFirstTest() **throws** InterruptedException {

*driver*.resetApp();

}

}

Here’s a brief rundown on what this all means

* //1 — Functions that have the @BeforeSuite annotation will be run before every test suite run. Such a function can be used to setup the application into the right state/condition before the individual test cases are run.
* //2 — We set the URL to the localhost where we will be running appium
* //3 — Setup an object that will hold the Appium [Capabilities](http://appium.io/docs/en/writing-running-appium/caps/)
* //4 — Initialise the driver with the URL and Capabilities object. This driver will be the main object used to interact with the device
* //5 — Functions that have the @AfterSuite annotation will be run after every test suite run. These functions should be used to clean up any changes done on the device before the next test suite run.
* //6 — Functions that start with @Test will be identified as TestNG tests. you can enable or disable a test in the test suite by toggling the value in (enabled=true). This test at the moment will just reset the application and no nothing else.

If you’ve set everything up correctly, you should be able to right click on the myFirstTest method, and *see* “Run As > 1 TestNG Test”. If you select that now you will see a bunch of errors in the console. That’s because we have yet to start Appium and connect our device.

**Setup To Run A Test On A Real Device**

Connect an Android device to your computer. On the Android phone, you’ll need to enable [USB Debugging](https://developer.android.com/studio/debug/dev-options). If the phone prompts you to Allow USB debugging for the computer that it’s connected to, allow it.

To save yourself a bit of headache you should set the device settings to sleep after 5 minutes of inactivity. This is to prevent the phone from locking before the script runs or during its run.

Check that the device is correctly connected by launching terminal and entering the following command:

adb devices

You should see one device connected like this:





Next, in terminal start the appium server with the following command

appium

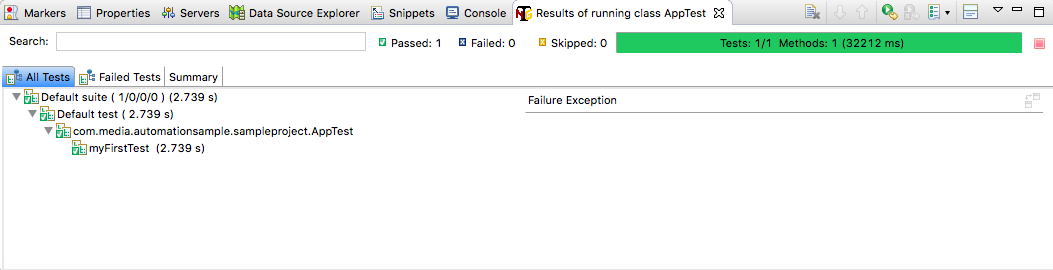
You should get some output that indicates the appium server is listening:





Return to your project in Eclipse and right click on the myFirstTest method, and select “Run As > 1 TestNG Test”. It’ll take a few minutes and you’ll see the device screen flicker a few times and hopefully… Voila!





Assuming you see the successful test case run, what test script did was install the application, restart it a couple times and then uninstall the application. Not very useful, but its working! Now let’s get to doing something productive with this script.

**Write An Actual Test**

The general flow of an appium automated test is to find an element and then either interact with it or assert something about it.

Let’s install the Material Dialogs APK to the device first to take a look at what we can test. Download it to your local system from [here](https://github.com/appium/sample-code/raw/master/sample-code/apps/ContactManager/ContactManager.apk) and install it into your device using adb

adb install [path-to-your-contact-manager-apk]

*e.g. adb install /Users/2359media/Downloads/ContactManager.apk*

Here is one flow from the application that we might want to test, written as a Gherkin story.

Given: Application is installed and launched

And: User is on the home page

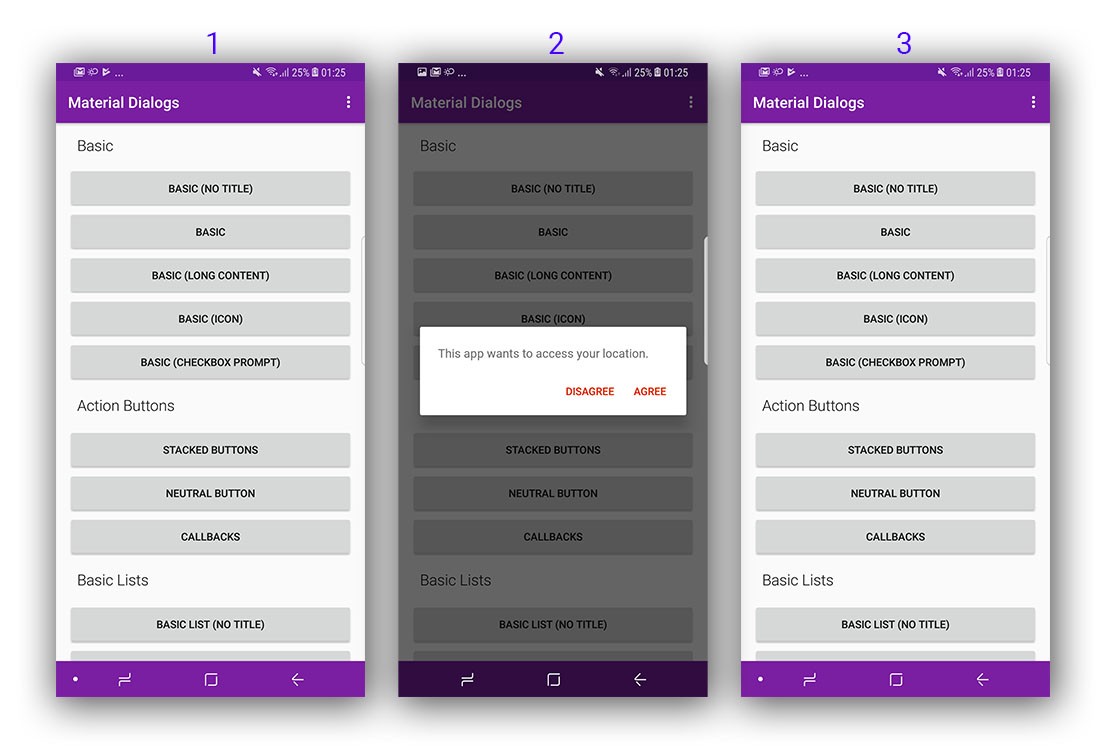
When: User selects button "BASIC (NO TITLE)"

Then: User should see popup with text "This app wants to access your location", "DISAGREE" and "AGREE"

When: User selects "AGREE"

Then: Popup should be dismissed





To identify elements, the Android SDK provides a tool called uiautomatorviewer. Launch Terminal and navigate to the location of the Android SDK and launch uiautomatorviewer with the following command:

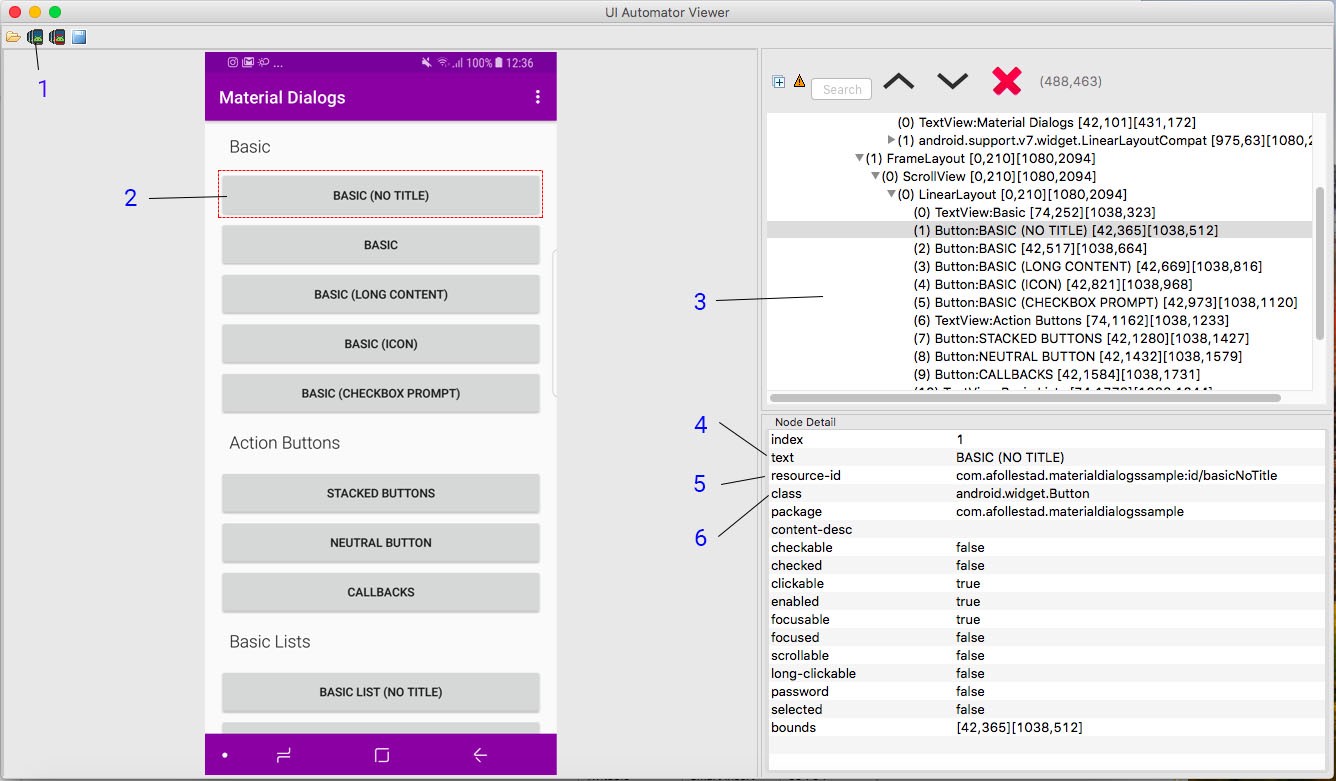
cd /Users/**2359media**/Library/Android/sdk/tools/bin

./uiautomatorviewer

(Be sure to change “2359media” to your own username)

Once the uiautomatorviewer is launched, connect your android device to the computer and launch the app on the Android device. Now click the device button on the top left of uiautomatorviewer to grab the view hierarchy, elements and screenshot.





UI Automator Viewer

* 1 — Device Screenshot (uiautomator dump) button. Clicking this will refresh UI Automator Viewer with the current view contents of the connected android device
* 2 — An interactive screenshot of the current screen on the Android device. Clicking on elements in the screenshot will change the contents of the right panels
* 3 — This is the view hierarchy, you can expand and collapse the nodes and select nodes to view their details in the panel below
* 4 — text: The text content of the selected element
* 5 — resource-id: The id of the selected element
* 6 — class: The class of the selected element

Using this tool, we can identify the elements that we need to write the following test case. Add this function to the AppTest class:

/\*

\* Test Name: testBasicNoTitle()

\* Given: Application is installed and launched

\* And: User is on the home page

\* When: User selects button "BASIC (NO TITLE)"

\* Then: User should see popup with text "This app wants to access your location", "DISAGREE" and "AGREE"

\* When: User selects "AGREE"

\* Then: Popup should be dismissed

\*/

@Test (enabled=**true**) **public** **void** testBasicNoTitle() **throws** InterruptedException {

// Find the button BASIC (NO TITLE) and click it

*driver*.findElementById("com.afollestad.materialdialogssample:id/basicNoTitle").click();

// Assert the presence of the popup title, AGREE and DISAGREE buttons

Boolean isTitlePresent = !*driver*.findElementsById("com.afollestad.materialdialogssample:id/md\_content").isEmpty();

Boolean isDisagreePresent = !*driver*.findElementsById("com.afollestad.materialdialogssample:id/md\_buttonDefaultNegative").isEmpty();

Boolean isAgreePresent = !*driver*.findElementsById("com.afollestad.materialdialogssample:id/md\_buttonDefaultPositive").isEmpty();

Assert.*assertTrue*(isTitlePresent && isDisagreePresent && isAgreePresent);

// Assert the contents of the popup title, AGREE and DISAGREE button

String popupTitle = *driver*.findElementById("com.afollestad.materialdialogssample:id/md\_content").getText();

Assert.*assertEquals*(popupTitle, "This app wants to access your location.");

String disagreeText = *driver*.findElementById("com.afollestad.materialdialogssample:id/md\_buttonDefaultNegative").getText();

Assert.*assertEquals*(disagreeText, "DISAGREE");

String agreeText = *driver*.findElementById("com.afollestad.materialdialogssample:id/md\_buttonDefaultPositive").getText();

Assert.*assertEquals*(agreeText, "AGREE");

// Click on the AGREE button

*driver*.findElementById("com.afollestad.materialdialogssample:id/md\_buttonDefaultPositive").click();

// Assert that the popup is no longer visible

Boolean isTitleStillPresent = !*driver*.findElementsById("com.afollestad.materialdialogssample:id/md\_content").isEmpty();

Assert.*assertFalse*(isTitleStillPresent);

}

Right click on the test code and run it to see it in action (don’t forget to start the appium server again if you previously stopped it).

**Running a set of test cases as a test suite**

Test cases should be as modular and granular as possible. Ideally each test case will test only test a specific individual function. So that when you run the test suite, you can view the report and identify which functions exactly have broken.

In the last part of this tutorial, we’re going to show you how to create a test suite to run all your test cases together.

Let’s add a second test function to

Next, in the Project Explorer, right click the project and click “New > File”. Create a file called testsuite.xml. Edit testsuite.xml and paste in the following contents.

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

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

<suite name=*"Automation Suite"* verbose=*"1"*>

<test name=*"All Testcases"*>

<packages>

<package name=*"****com.media.automationsample.sampleproject****">*</package>

</packages>

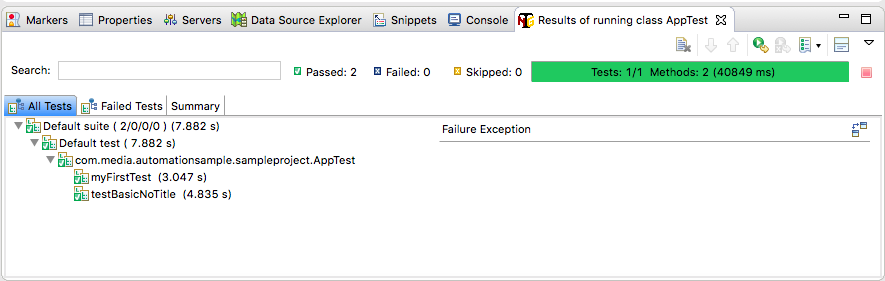
</test>

</suite>

Be sure to replace your package name accordingly

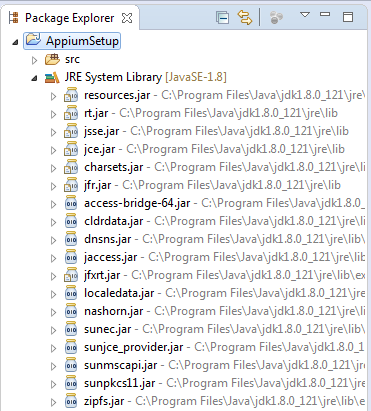
Save the file and right click anywhere in the file contents, select “Run As > TestNG Suite”. In the Eclipse test report you should see 2 successful test runs.





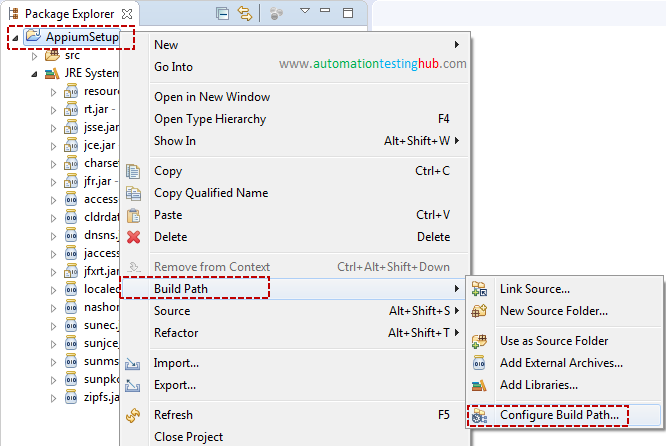
**Add Appium JAR files to your Eclipse Project**

Expand ‘AppiumSetup’ project. You will see that it has 2 sub-folders or components inside it. These sub-folders are: **src** and **JRE System Library**. If you expand JRE System Library, you will see that it has some JAR files already added as shown in the below image. This is the minimum set of JAR files that is required to run a basic Java program.

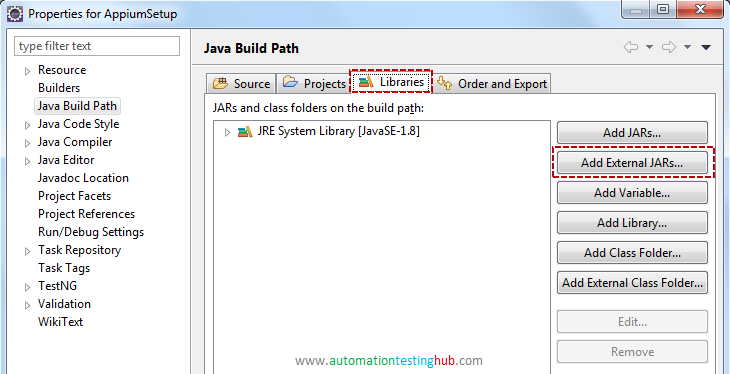


Since you would be creating and running Appium scripts, you would need to add the Appium specific JAR files as well. Follow the steps given below to add these JAR files to your Appium project –

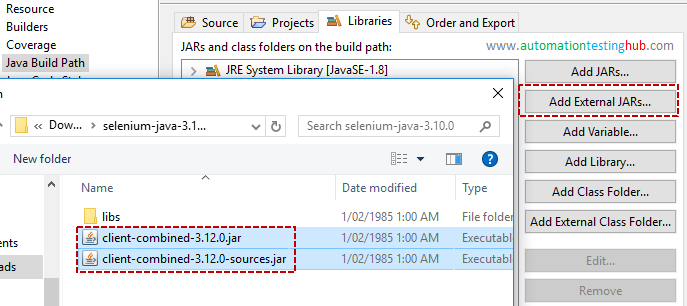
**1.**Right click on the Project. Then select **Build Path > Configure Build Path…** option



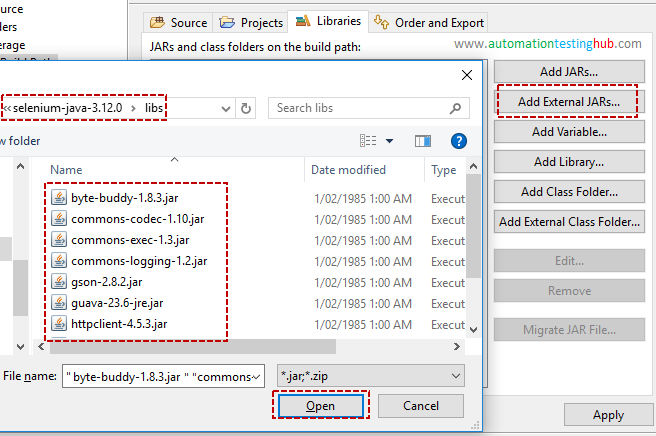
**2.**Properties window will get opened as shown below. From the **Libraries** tab, click on **Add External JARS…** button.



**3.**Now navigate to the folder where you had downloaded Selenium WebDriver JAR files. Select the selenium JAR (named client-combined…) and click on Open button

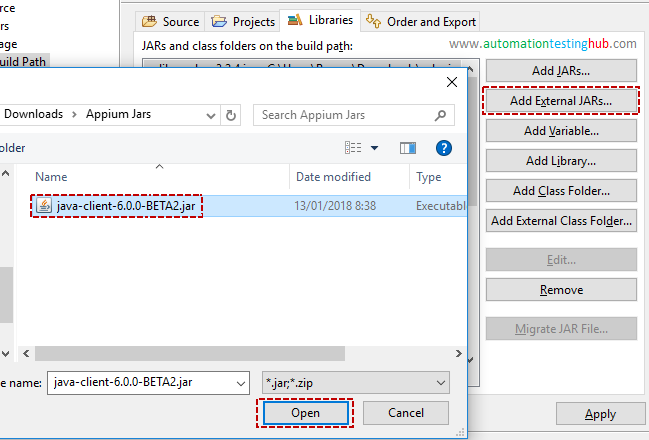


Do the same for all the JARs in the lib folder



With this, you have now added all the Selenium JARs to the project. Now let us add the Appium Java Client jar file.

**4.**Click on Add External JARs… button, and navigate to the folder where you have saved Java Client JAR. Select the JAR and click on Open button



Links:

**TestNG Manual:**

<https://www.toolsqa.com/testng-tutorial/>

<https://stackoverflow.com/questions/37054269/what-exactly-does-a-testng-thread-count-do>

**Appium waits:**

<https://appiumpro.com/editions/21>

<http://www.qaautomated.com/2016/11/implicit-explicit-wait-with-appium.html>

<https://www.guru99.com/implicit-explicit-waits-selenium.html>

Related issues:

<https://stackoverflow.com/questions/53257500/android-appium-implicit-wait-on-androiddriver-appium-doesnt-work>

**Run tests from XML file:**

<https://www.softwaretestingmaterial.com/create-testng-xml-file/>

**SendKeys methods:**

<https://chercher.tech/java/sendkeys-click-clear-submit-selenium-webdriver>

**Get ADR and iOS elements names/IDs:**

<https://www.youtube.com/watch?v=LT3mtdCm_vw>

<https://www.youtube.com/watch?v=RagNvVuCvLI>

**How to create .ipa for Appium:**

<https://wiki.genexus.com/commwiki/servlet/wiki?34616,HowTo%3A+Create+an+.ipa+file+from+XCode>

**Locators:**

<https://saucelabs.com/blog/advanced-locator-strategies>

<https://seleniumbycharan.com/2016/08/07/finding-elements-using-locators-in-appium/>

**Java Vars:**

<https://www.javatpoint.com/java-variables>

<https://www.learnjavaonline.org/en/Variables_and_Types>

<https://www.w3schools.com/java/java_variables.asp>

**Java overriding:**

<https://www.javatpoint.com/method-overriding-in-java>

**Val in Java:**

<https://projectlombok.org/features/val>

**XPath:**

<https://chercher.tech/java/relative-xpath-selenium-webdriver>