Getting Started with Appium

By Ken Asanion

For any questions or clarifications regarding the instructions I provided, contact me at kenasanion@gmail.com

Disclaimer: I'm currently an OSX user, so there might be some lapses regarding the steps that involves Windows such as environmental variables. I've set up a guide for it. We will use Eclipse as an example for our exercises moving forward.

Introduction

Usually, setting your test automation environment is a bittersweet process. You need to install a lot of things for things to work smoothly. Sometimes, it is one of the major problems where scripters lag all the time. But once you get it working, test automation can be easy and fun. Test automation can also be a fragile thing to do. If you're left hanging on the bad practice of scripting, then the scripts that you are doing since day 1 can stall down your progress rather than making it fast and cost efficient. From this section forward, I'll use the best practices that I believe will make our progress safe and sound. I'd also appreciate it if you have any feedback that you may think of under the sun. Without further ado, let's get started!

Helpful Links

Official Sample Codes: https://github.com/appium/sample-code

Setting up Appium in Eclipse

Prerequisite: Make sure you properly installed everything before proceeding.

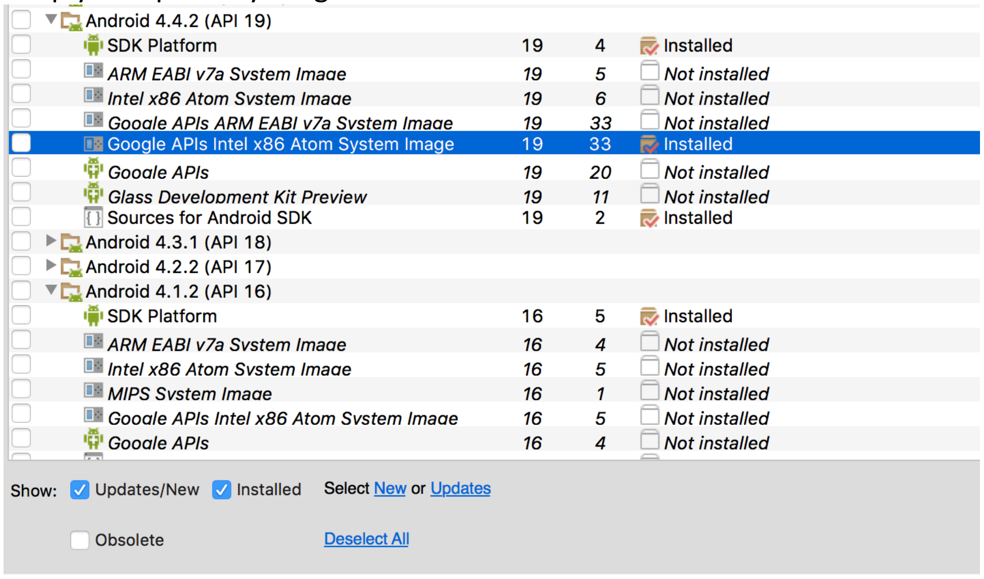
1. Open Eclipse IDE that we have just installed.
2. Install ADT Plugin. This would help us to access our SDK tools with ease.
    - Follow the official steps here https://stuff.mit.edu/afs/sipb/project/android/docs/sdk/installing/installing-adt.html
3. You may now access Android SDK Manager and Virtual Device Manager through Eclipse.

Setting up Android Emulator

Before creating our first script, we need to ensure that our runtime environment is ready. For this training, we would be using the Android SDK Manager and Virtual Device Manager. To access it, locate your SDK tools in your machine or if you have ADT tools plugin, you can access it via the following:

In order for us to create a virtual device, we need to verify if there are images available. To do that follow the steps below.

1. Open Android SDK Manager.
2. Locate the Android version that you are looking for and collapse it.
3. Check if you have an ARM or Intel images available. To know more of the difference between the two you can check this link: https://www.androidauthority.com/arm-vs-x86-key-differences-explained-568718/
4. If you have an ARM or intel images installed, skip this step. Otherwise, install an image of your choice. On my part, I'm choosing the Intel x86 Atom System Image. Tick the checkbox beside it and click Install Package. You would be prompted a license check. Simply accept everything.



Android 4.4.2 (API 19)			
SDK Platform	19	4	Installed
ARM EABI v7a System Image	*19*	*5*	*Not installed*
Intel x86 Atom System Image	*19*	*6*	*Not installed*
Google APIs ARM EABI v7a System Image	*19*	*33*	*Not installed*
Google APIs Intel x86 Atom System Image	19	33	Installed
Google APIs	*19*	*20*	*Not installed*
Glass Development Kit Preview	*19*	*11*	*Not installed*
Sources for Android SDK	19	2	Installed
Android 4.3.1 (API 18)			
Android 4.2.2 (API 17)			
Android 4.1.2 (API 16)			
SDK Platform	16	5	Installed
ARM EABI v7a System Image	*16*	*4*	*Not installed*
Intel x86 Atom System Image	*16*	*5*	*Not installed*
MIPS System Image	*16*	*1*	*Not installed*
Google APIs Intel x86 Atom System Image	*16*	*5*	*Not installed*
Google APIs	*16*	*4*	*Not installed*

Show: [✓ Updates/New] [✓ Installed] Select New or Updates

Obsolete Deselect All

* Images are heavy in your machine's space. Consider downloading only some that you will really use.

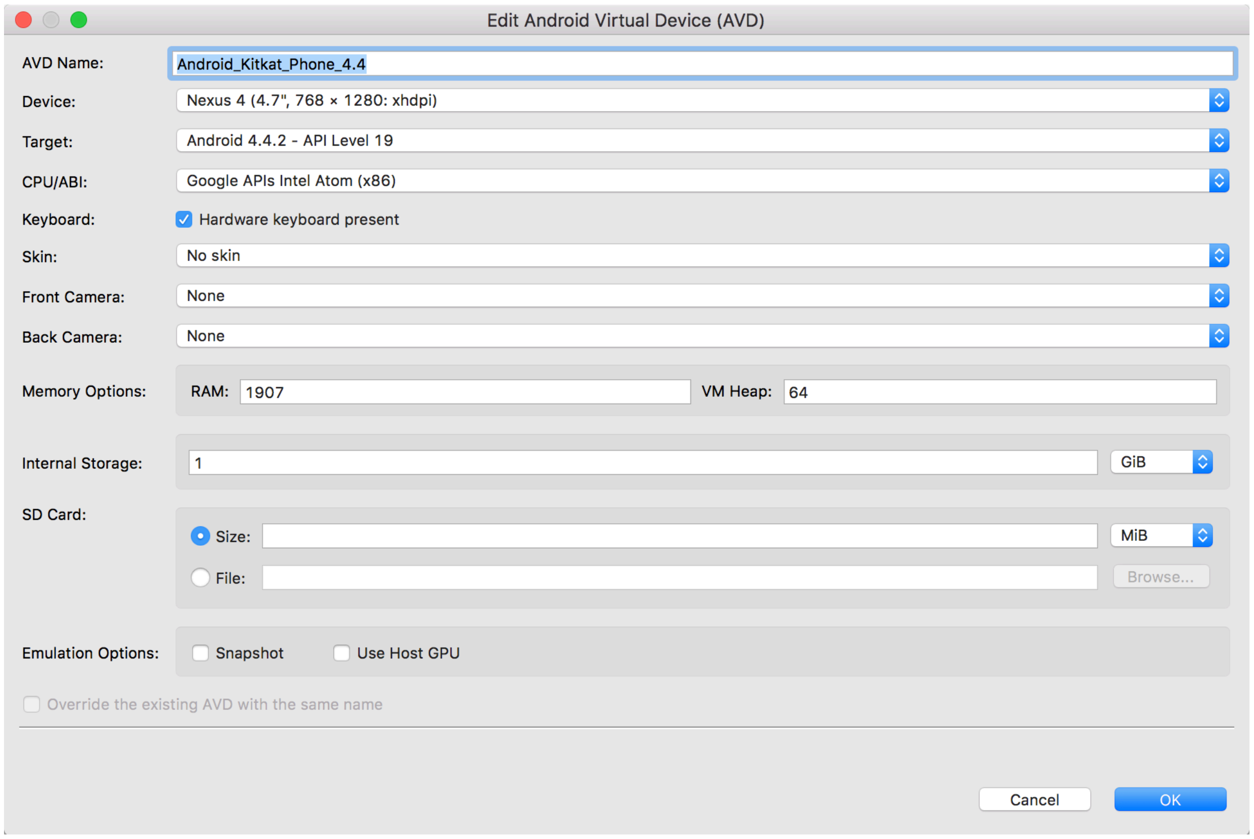
5. Your download should start now and you should see the progress at the bottom of the SDK Manager window.
6. Once finished, close it and **restart Eclipse** (very important step).

If by any chance that the SDK Manager have other packages that are selected, you may opt to **Deselect All** and then download only the image. This would save us time by not downloading everything.

To greatly enhance your emulator's speed for Intel images, you may need to download the Intel x86 Emulator Accelerator which can be accessed under 'Extras' category. Do note that this is not compatible on everyone! The SDK Manager would prompt you if it is not.

Creating and Running the Android Emulator

1. Locate and open Android Virtual Device Manager.
2. Click create and a new window will appear.
3. Follow the settings as shown below:



AVD Name:	Android_Kitkat_Phone_4.4
Device:	Nexus 4 (4.7", 768 × 1280: xhdpi)
Target:	Android 4.4.2 - API Level 19
CPU/ABI:	Google APIs Intel Atom (x86)
Keyboard:	[✓] Hardware keyboard present
Skin:	No skin
Front Camera:	None
Back Camera:	None
Memory Options:	RAM: 1907 VM Heap: 64
Internal Storage:	1 GiB
SD Card:	[✓ Size:] MiB / File:
Emulation Options:	☐ Snapshot ☐ Use Host GPU

☐ Override the existing AVD with the same name

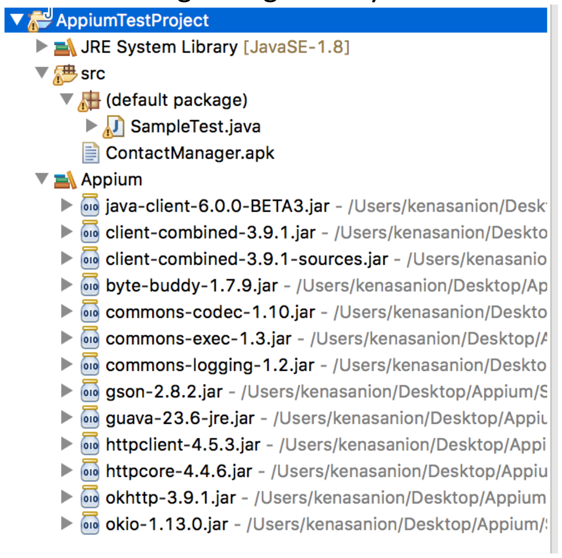
* You may search on what does each field mean. But for now, this should work.
* If there are no images located in CPU/ABI dropdown field, you need to check if you downloaded the right images for Android 4.4.2 in the SDK Manager. Also, make sure that you **restarted eclipse** before proceeding on this step.
* AVD devices consume a lot of your machine's space. The higher the internal storage you allocate, the higher it would be sitting in your machine.

4. Click ok and verify that your AVD device was successfully created. If it does, it should show in your AVD manager.
5. Click start and your emulator should be up and running.

*If your adb is configured properly, type in `adb devices` to check if your device was successfully attached. To know more what you can do with **adb** CLI command, you can check on this link https://developer.android.com/studio/command-line/adb.html

Creating our first test automation project

1. On eclipse, create a new project under File > New > Java Project
2. Type in a name for your project and leave the default setting and choose finish.
3. You would be directed to the Eclipse workspace window.
4. Navigate to package explorer
5. Create a User Library that we could reuse for our Appium projects.
    - Go to Project Properties by right clicking project in the package explorer > Build Path > Configure Build Path…
    - Go to Libraries Tab
    - Click the Add Library Button
    - Select User Library and then click next
    - Click User Libraries
    - Click New and type in the name ‘Appium’ or anything you want.
    - Click Ok
    - You shall now see your newly created library. Lets add the jar files by clicking ‘Add External JARs’
    - Locate the Appium jar and selenium files you downloaded and add every jar files. Import the following:
        I. Appium Java Client jar
        II. Selenium
            - client combined jar files
            - libs jar files
    - You should now see the jar files under the ‘Appium’ category.
    - Apply and close remaining dialogs after you are done.

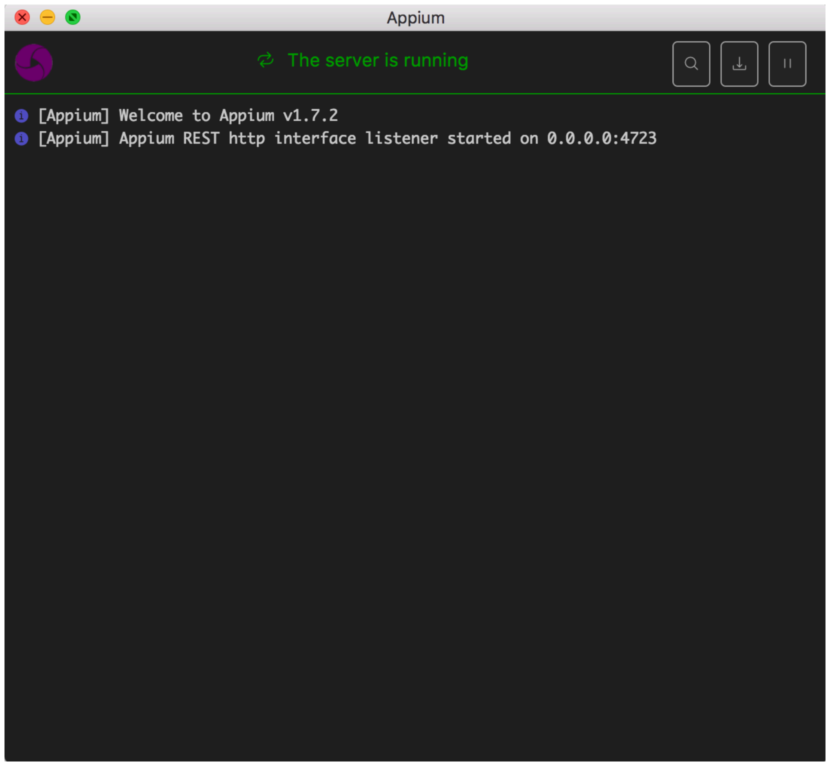


```
▼ AppiumTestProject
  ▶ JRE System Library [JavaSE-1.8]
  ▼ src
    ▼ (default package)
      ▶ SampleTest.java
      ContactManager.apk
  ▼ Appium
    ▶ java-client-6.0.0-BETA3.jar - /Users/kenasanion/Desk
    ▶ client-combined-3.9.1.jar - /Users/kenasanion/Deskto
    ▶ client-combined-3.9.1-sources.jar - /Users/kenasanio
    ▶ byte-buddy-1.7.9.jar - /Users/kenasanion/Desktop/Ap
    ▶ commons-codec-1.10.jar - /Users/kenasanion/Deskto
    ▶ commons-exec-1.3.jar - /Users/kenasanion/Desktop/
    ▶ commons-logging-1.2.jar - /Users/kenasanion/Deskto
    ▶ gson-2.8.2.jar - /Users/kenasanion/Desktop/Appium/
    ▶ guava-23.6-jre.jar - /Users/kenasanion/Desktop/Appi
    ▶ httpclient-4.5.3.jar - /Users/kenasanion/Desktop/Appi
    ▶ httpcore-4.4.6.jar - /Users/kenasanion/Desktop/Appiu
    ▶ okhttp-3.9.1.jar - /Users/kenasanion/Desktop/Appium
    ▶ okio-1.13.0.jar - /Users/kenasanion/Desktop/Appium/
```

Your workspace in the package explorer should now look like this. You’re done! Now we’re ready to run Appium and code our very first test script!

Launching Appium Server

1. Launch Appium Server
2. Click `Start Server v{version.number}` using the default host of 0.0.0.0 and port 4723.
3. Verify that the server is running and running. This server would then allow your scripts to communicate with the device through HTTP calls.



```
[Appium] Welcome to Appium v1.7.2
[Appium] Appium REST http interface listener started on 0.0.0.0:4723
```

Writing our first Appium test script

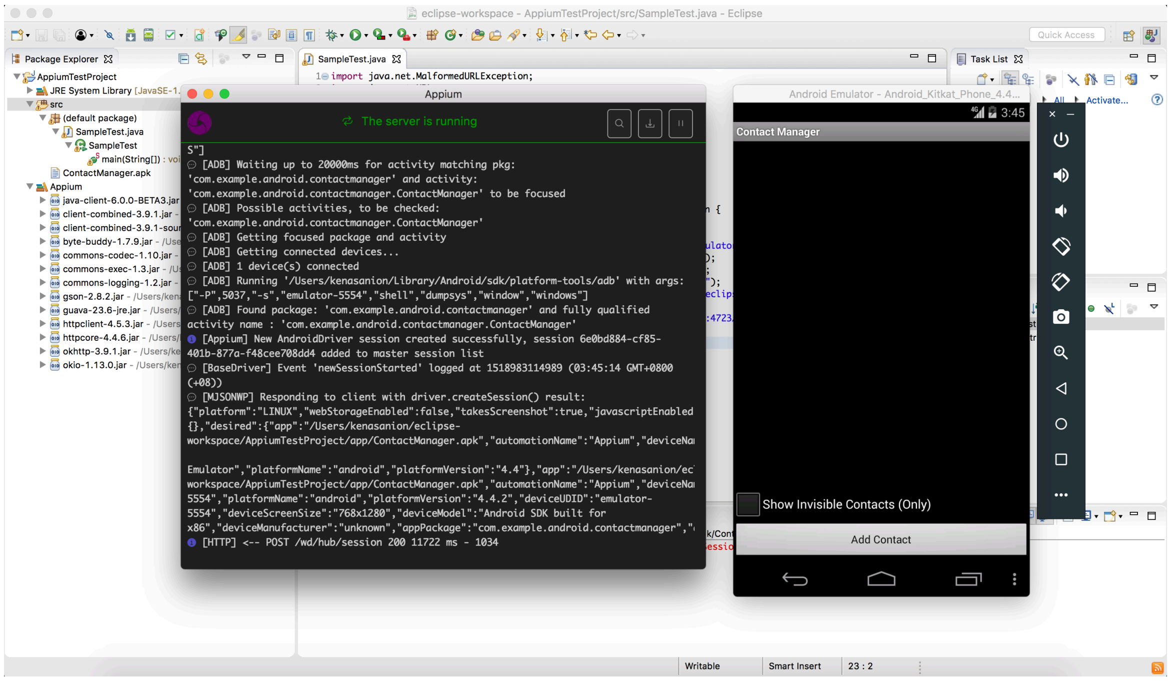
1. Go back to the project we created and create a new file. Typically, we normally place our source codes under src folder. Right click src folder and go to New > Class
2. Type in a name for your new java class and click Finish. By convention, we should end our test scripts using the word *Test* e.g. MobileTest, ShoppingTest, BankTest, etc.
3. A new file should appear. This is the default structure of a Java class file. Before we dive in too deep, let us first try to copy and paste the code below. Parts of these section would be explained in the demo.

```
import java.net.MalformedURLException;
import java.net.URL;

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

import io.appium.java_client.MobileElement;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.remote.MobileCapabilityType;

public class SampleTest {
       public static void main(String[] args) throws MalformedURLException {
              DesiredCapabilities dc = new DesiredCapabilities();

              dc.setCapability(MobileCapabilityType.DEVICE_NAME, "Android Emulator");
              dc.setCapability(MobileCapabilityType.PLATFORM_NAME, "Android");
              dc.setCapability(MobileCapabilityType.PLATFORM_VERSION, "4.4");
              dc.setCapability(MobileCapabilityType.AUTOMATION_NAME, "Appium");
              dc.setCapability(MobileCapabilityType.APP, "/PATH/TO/APK");

              URL remoteURL = new URL("http://0.0.0.0:4723/wd/hub");

              AndroidDriver<MobileElement> driver = new
AndroidDriver<MobileElement>(remoteURL, dc);

       }
}
```

4. Run the script in eclipse by clicking the ▶ on the upper tab.

- ✓ Before running this code, ensure that your Appium server is running as well as your Android KitKat Emulator.
- ✓ Make sure you setup ANDROID_HOME correctly. Otherwise, you would encounter an error. Also ensure that you restart Eclipse and Appium server after setting ANDROID_HOME in your system variables.

If everything went right, you would notice that the server receives and sends request to the emulator. Viola! The app you told your script to install was successfully installed to your emulator and it automatically ran. Now let's try to add a simple click functionality.


eclipse-workspace - AppiumTestProject/src/SampleTest.java - Eclipse
Package Explorer
AppiumTestProject
SampleTest.java
Appium
The server is running
Android Emulator - Android_Kitkat_Phone_4.4...
Contact Manager
Show Invisible Contacts (Only)
Add Contact
Task List
Writable
Smart Insert
23 : 2

```
S"]
[ADB] Waiting up to 20000ms for activity matching pkg:
'com.example.android.contactmanager' and activity:
'com.example.android.contactmanager.ContactManager' to be focused
[ADB] Possible activities, to be checked:
'com.example.android.contactmanager.ContactManager'
[ADB] Getting focused package and activity
[ADB] Getting connected devices...
[ADB] 1 device(s) connected
[ADB] Running '/Users/kenasanion/Library/Android/sdk/platform-tools/adb' with args:
["-P",5037,"-s","emulator-5554","shell","dumpsys","window","windows"]
[ADB] Found package: 'com.example.android.contactmanager' and fully qualified
activity name : 'com.example.android.contactmanager.ContactManager'
[Appium] New AndroidDriver session created successfully, session 6e0bd884-cf85-
401b-877a-f48cee708dd4 added to master session list
[BaseDriver] Event 'newSessionStarted' logged at 1518983114989 (03:45:14 GMT+0800
(+08))
[MJSONWP] Responding to client with driver.createSession() result:
{"platform":"LINUX","webStorageEnabled":false,"takesScreenshot":true,"javascriptEnabled
{},"desired":{"app":"/Users/kenasanion/eclipse-
workspace/AppiumTestProject/app/ContactManager.apk","automationName":"Appium","deviceNa

Emulator","platformName":"android","platformVersion":"4.4"},"app":"/Users/kenasanion/ec
workspace/AppiumTestProject/app/ContactManager.apk","automationName":"Appium","deviceNa
5554","platformName":"android","platformVersion":"4.4.2","deviceUDID":"emulator-
5554","deviceScreenSize":"768x1280","deviceModel":"Android SDK built for
x86","deviceManufacturer":"unknown","appPackage":"com.example.android.contactmanager","
[HTTP] <-- POST /wd/hub/session 200 11722 ms - 1034
```

Adding a simple gesture to the button

1. Go back to your source code.
2. Add this one-line of code just below AndroidDriver. Be wary of the braces.

   ```
   driver.findElement(By.Id("Add Contact")).click();
   ```

3. Run the script.

That's it! It's that simple to automate using Appium. Now, I think we're ready to handle some technical and advanced stuff from this point onwards ☺

If you have any other queries feel free to reach me out through email.