**Why Appium?**

**Robotium** – only for Android Devices

**Calabash** – only for IOS Devices

**Selenium Android** – only for Android Devices

**Appium – for both Android and IOS Devices (Only one framework that supports Cross Platform Automation)**

**Supporting Languages:** Java, C#, JavaScript, Python, Ruby

**Appium Architecture:**

Diagram

Description automatically generated

**Why we need Node JS when we work with Appium?**

Appium server is written in JS. So, to install and run the Appium server we need Node JS.

**Steps to configure Appium in Windows:**

Download JAVA

Download Android Studio and Findout the SDK Path

Download Node Js

Set (Java, Android SDK, Node JS) Home Paths in System Variables

Open Android Studio and Configure Emulator / Virtual Device

Install and start the Appium Driver via Node

Install Eclipse Editor

Understand the desired capabilities to setup the Environment in the Appium

Appium first program to invoke the Android Apps.

Text, letter

Description automatically generated

Text, letter

Description automatically generated

Text

Description automatically generated with medium confidence

**UIAutomator:**

UIAutomator is UI testing framework introduced by google to facilitate automation on a Androi Device or Emulator.

Appium leverages the UIAutomator and wrapped up the same as a UIAutomator2 Driver. By using the **UIAutomator2 driver** we can perform the UI Automation testing.

**To start the Appium server:**

$appium

**To list down the drivers for the Appium:**

Text

Description automatically generated

**To install the driver:**

$appium driver install DRIVER\_NAME

Text

Description automatically generated

**By using UiAutomator2Options class we can create an Options to access the App:**

UiAutomator2Options capabilities = new UiAutomator2Options();

capabilities.setDevice(“DEVICE\_NAME”);

capabilities.setApp(“PATH\_TO\_APK\_FILE”);

**Appium Inspector:**

To identify the locators we are using the Appium Inspector.

How to configure? [To specify the correct APP at correct Mobile Device]

app: “PATH\_TO\_APK\_FILE”,

deviceName: “NAME\_OF\_THE\_DEVICE”

platformName: “android/ ios” // Since, this **Appium inspector** can used for **iOS** as well. We need to provide this.

automationName: “UIAutomator2”

Graphical user interface, application

Description automatically generated

**Appium Locators:**

Under class **By**:

Xpath, id

Under class **AppiumBy:**

accesibilityId, androidUIAutomator

**Appium Gestures (like Double Click, Long Press):**  
[**https://appium.io/docs/en/writing-running-appium/android/android-mobile-gestures/**](https://appium.io/docs/en/writing-running-appium/android/android-mobile-gestures/)

To create an Long Press we need to pass some JAVA SCRIPT into appium. To pass that JavaScript we have an method called **executeScript(event, Where to perform specific event as a Map<Key, Values>).**

Text

Description automatically generated

**UiScrollable:**

<https://developer.android.com/reference/androidx/test/uiautomator/UiScrollable>

**We can open the App using Package and Activity:**

Package – indicates the specific App

Activity – indicates the specific page of the App

So that, we can directly launch the Page which we need without opening the App from the initial stage.

**How to find ‘Package’ and ‘Activity’ ?**

$adb shell dumpsys window | grep -E ‘mCurrentFocus’ [For MAC]

$adb shell dumpsys window | find “mCurrentFocus” [For Windows]

Above command will give the result as **<Package\_Name>/<Activity\_Name>**

**How to implement ?**

Activity activity = new Activity(<Package\_Name>, <Activity\_Name>);

Driver.startActivity(activity);