SK Institute

**APPIUM**

*by*

Srikanth Pittala Sr. Automation Engineer

**What is Appium?**

* It is an automation tool used to automate & test mobile applications (Native app, mobile web app, Built-in app & hybrid app) on both Android and IOS platform.
* It written in Node.js
* It is developed by Sauce Labs company/Selenium community.
* It is open-source mobile automation tool.

**Appium architecture.**



* When user (Appium client) sent a request (consists Desired Capabilities i.e., package name, activity name, platform name & version, application name) through JSON protocol to Appium server It will create a session id and sent it back as response to client.
* Based on this session id client (Selenium) and server (Appium) will communicate each other.
* Appium server creates service request and send it to UI Automation (IOS) or UI Automator (Android).
* These UI Automator or Automation will communicate with Bootstrap.jar which is running in emulator, simulator & real device and convert request into operations.

**How does it work**?

Appium test script written in IDE will interact with the Appium Server which is nothing but the node server with the specified IP address and port number. Node server again passes the request to mobile devices or emulators using the uiautomator or XCUITest as a JSON format. All the UI elements associated with the mobile application can be controlled by using the appium client which is nothing but the derived one of selenium.

**Note**: Remember using Windows only Android can be automated while in iOS both Android and iOS can be automated.

**Advantages:**

* It supports all languages
* Platform independent (android, IOS & Windows)
* Supports all application i.e., Built-in apps, native apps, web apps & hybrid apps.
* Open source

**Disadvantages:**

* Server should be up and running before executing script.
* Slow execution on virtual devices
* Supports only android API levels which are greater than 17.
* Parallel execution will not support on single device.
* Can’t handle Toast messages

**Driver-Specific Setup:**

You probably want to use Appium to automate something specific, like an iOS or Android application. Support for the automation of a particular platform is provided by an Appium "driver". There are several such drivers that give you access to different kinds of automation technologies, and each come with their own particular setup requirements. Most of these requirements are the same requirements as for app development on a specific platform.

* iOS 9.3 and above: Apple's XCUITest
* iOS 9.3 and lower: Apple's UIAutomation
* Android 4.3+: Google's UiAutomator/UiAutomator2(default)
* Windows: Microsoft's WinAppDriver

**Emulator & Simulator:**

* These are the virtual devices which have same functionality of real devices except camera.
* A virtual device (android or ios) on windows os is called emulator and virtual device (android or ios) on IOS os is called simulator.

**Different types of mobile applications:**

1. **Built-in application or System applications.**

Applications which are installed as part of operation system are called built-in applications.

We can’t un-install these applications from mobile, but we can disable or enable it.

**Example**: Camera, Calculator, Alarm, Calendar etc.

1. **Native application:**

Applications which are developed on base of particular platform (Android & IOS) are called native applications.

**Example**: WhatsApp, Facebook, Gmail etc.

1. **Mobile web application:**

These are web applications or websites which are opened with the help of any browser in mobile.

**Example**: Chrome Browser, Firefox Browser etc.

1. **Hybrid application:**

These are combination of Native and Web applications.

Example: Amazon, Flipkart, Facebook, Instagram, Daily hunt etc.

**Different types of mobile Operating Systems:**

1. Android OS by Google
2. IOS OS by Macintosh
3. Windows by Microsoft

**List of Android versions.**

**Get sample apk for testing**

[**https://appium.io/docs/en/about-appium/getting-started/index.html**](https://appium.io/docs/en/about-appium/getting-started/index.html)

[**https://practice-project**](https://www.rahulshettyacademy.com/practice-project)

**Installations & Setup:**

1. Install Java 8 and setup environment variable.
2. Install Nodejs and setup environment variable.
3. Install Maven and setup environment variable.
4. Install Android studio for android SDK packages.
5. Setup android environment variable.
6. Creating an emulator via android studio.
7. Enabling Developer mode option in android phone.
8. Install PDANet (universal driver) to connect with Android devices.
9. Install Xcode, HomeBrew, Carthage, appium-xcuitest-driver for (Mac only).
10. Install Appium via npm.
11. Install Appium via installer.
12. Install Appium Doctor.

**For Windows:**

For best ref:

<https://support.smartbear.com/testcomplete/docs/app-testing/mobile/device-cloud/configure-appium/android-on-windows.html>

1. Install NodeJS (latest/8.11.0)

* When we install node.js by default we get 2 softwares node.js and npm
* Npm stands for node package manager, it is a command line tool which is used to install and operate node.js applications ie. (Protractor, appium, ect)
* **npm -v**

1. Android Studio (Required java (1.8) pre-installed)
2. Appium Desktop application (v1.22.3-4)

* Appium server is developed by Node.js so it required node.js platform to run.
* Appium can be installed in one of two ways: via NPM or by downloading Appium Desktop, which is a graphical, desktop-based way to launch the Appium server.

1. Install Appium Doctor

npm install -g appium-doctor

1. Jar files:

* Selenium standalone jar files (selenium-server-standalone-4.3.0)
* Appium client server jar files (java-client-7.5.1)
* Common-io (commons-io-2.11.0)
* Common lang3 (3.12.0)

1. Environment variable

System Variables

* JAVA\_HOME
* ANDROID\_HOME (C:\Users\admin\AppData\Local\Android\Sdk)
* NODE\_HOME (C:\Program Files\nodejs)

Edit path & add

* % JAVA\_HOME%/bin
* % ANDROID\_HOME%/platform-tools
* % ANDROID\_HOME%/tools
* % ANDROID\_HOME%/tools/bin
* % ANDROID\_HOME%/build-tools
* C:\Program Files\Android\Android Studio\bin
* C:\Program Files\nodejs\node\_modules\npm\bin

**Installations & Setup on Mac:**

1. Install Brew
2. Install Java (1.8)

To install Java open terminal “brew install openjdk@8”

1. Install NodeJS and npm
2. Install carthage.

brew install carthage

1. Install Appium

npm install -g appium

1. Install appium doctor (To verify that all of Appium's dependencies)

npm install appium-doctor -g

1. Install android Studio

**Environment variable**

Step1: open terminal and type: vi ~/.bash\_profile

Step2: click on i to insert bash\_profile

Step3: enter variables

Step4: save and quit bash\_profile

ESC + wq

Step5: To save variable permanently

source ~/.bash\_profile

Note: in mac no need to configure node,python..etc variable because by default node js executable filed are dump into /usr/local/bin folder.

**What is APK?**

* APK stands for Android Package Kit and it's the file format that Android uses distribute and install apps.
* APK is an archive file contain all the elements that an app needs to install correctly on android device.

**What is IPA?**

* IPA stands for iOS App Store Package and it's the file format that IOS uses distribute and install apps.
* IPA is an archive file contain all the elements that an app needs to install correctly on IOS device.

**What is ADB:**

ADB stands for Android debugging bridge.it is a command line tool which is used to communicate with virtual and real devices connected through USB or WI-FI.

**Advantages:**

1. We can get all devices information of virtual and Real devices (connected with USB or WIFI).

**Command line:** adb devices

**For IOS**:

**Command line:** xcrun simctl list

1. We can connect real device with WIFI

**Command line:** adb tcpip 5555

adb connect <mobile ip>:5555

1. We can get all packages information

**Command line:** adb shell pm list packages

1. We can get specific apk from device

**Command line:** adb shell pm list packages

adb shell pm path <package-name>

adb pull <package-location> <path-on-computer-to-store-APK>

1. We can install APK on device

**Command line:** adb install <apk path>

1. We can kill adb server

**Command line**: adb kill-server

1. We can restart adb server

**Command line**: adb start-server

1. Open an application with app package and app activity

**Command line:** adb shell am start -n packageName/AppActivity

**Example:**

adb shell am start -n com.android.chrome/com.google.android.apps.chrome.Main

**What is xcrun**:

* xcrun is a command line tool that helps managing Xcode versions on your mac system

**Advantages**:

1. We can get all devices information of virtual and Real devices (connected with USB or WIFI).

**Command line**: xcrun simctl list

1. We can install IPA on device

**Command line**: xcrun simctl install <simulator-id> <ipa-path>

**App Package:**

* Application package identifier to be started. If not provided, then UiAutomator2 will try to detect it automatically from the package provided by the app capability.
* Simple app package is called as application name
* Package consists collection of all functionalities/app activities.
* Every application consis only one package name

**App Activity:**

* Main application activity identifier. If not provided, then UiAutomator2 will try to detect it automatically from the package provided by the app capability.
* Each and every funtinality in an application we have an app activity name.
* We can directly open page/funtionality of an application with app activity.
* Every application consists multiple acitivities

**How to find App Package and Activity?**

**Method: 1 (Through adb command)**

**For ref:**

[**https://support.testsigma.com/support/solutions/articles/32000019977-how-to-find-app-package-and-app-activity-of-your-android-app**](https://support.testsigma.com/support/solutions/articles/32000019977-how-to-find-app-package-and-app-activity-of-your-android-app)

1. Firstly, connect your Android device or emulator to the PC/Mac and open the App whose details you want to inspect i.e Whatsapp.
2. Open a Command Prompt or Terminal window and use ‘adb devices‘ command to see the list of connected devices. A list of Android devices connected to the computer is displayed along with their device IDs.
3. Finally, type the below-given command to get the information about the currently open application, i.e in our case, WhatsApp.

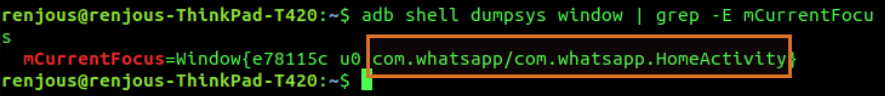
**For Mac/Linux**:

adb shell dumpsys window | grep -E 'mCurrentFocus'

**For Windows**:

adb shell dumpsys window | find "mCurrentFocus"

The Android Application Package name and Application Activity name are shown as given below:



Here, the part before the '/' character i.e com.whatsapp is the Package name and the part after that i.e com.whatsapp.HomeActivity is the Activity name.

**Note**: We need to make sure that the WhatsApp application is opened on the device and device is not locked before the procedure. We can use the same method for finding the details of any Application installed on the device.

**Method-2: (Through Appium inspector)**

Step-1: Create app session in appium inspector

Step-2: Click on Actions Tab

Step-3: Select Device >> Android Activity

Step-4: Collect app activity and app package by clicking on Current Activity & Current Package buttons

**Graphical user interface, text, application, email

Description automatically generated**

**Method-3: (using appium code)**

1. getCurrentPackage(): Get the name of the current Android package

**Example**:

String package = driver.getCurrentPackage();

1. currentActivity(): Get the name of the current Android activity

**Example**:

String activity = driver.currentActivity();

**Method-4: (using appium server log code)**

After installing app on devices appium server log will show package and activity

**What is bundle ID in iOS app**?

* The bundle-Ids resource represents the app's unique identifier that you can register, modify, and delete. You need a bundle ID before you can assign capabilities with the Bundle ID Capabilities resource or create a provisioning profile with the Profiles resource.

**How to get bundle-id to launch installed iOS app in simulator**:

* right click you downloaded .app file and select "show package contents" option
* info.plist double click this file
* get "bundle identifier" field value and pass to capabilities

**What is AVD Manager?**

AVD stands for Android virtual device.

AVD Manager is a platform present in android studio that helps to create and manage android virtual devices.

**What is SDK Manager?**

The SDK manager is a command line tool that allows you to view, install, update, and uninstall packages for the Android SDK

**Appium Inspector:**

Graphical user interface, application, Word

Description automatically generated

* **What Is Appium Inspector?**
  + Appium inspection is a standard procedure to identify the UI elements of a mobile app uniquely. It works with both real devices or simulators(iOS) or emulators (**Android**).
  + Appium inspector is a combination of the Appium server itself and the inspector, which is designed to help you discover all the visible elements of your app on both android and Ios devices.
  + When you click the app to open, it will start the Appium server with the default IP address set to 0.0.0.0. and the port to be used in 4723.
* **Record and Play using Appium Inspector**
* **Locating elements with the help of Appium Inspector**

### **Inspecting Page elements using Appium inspector**

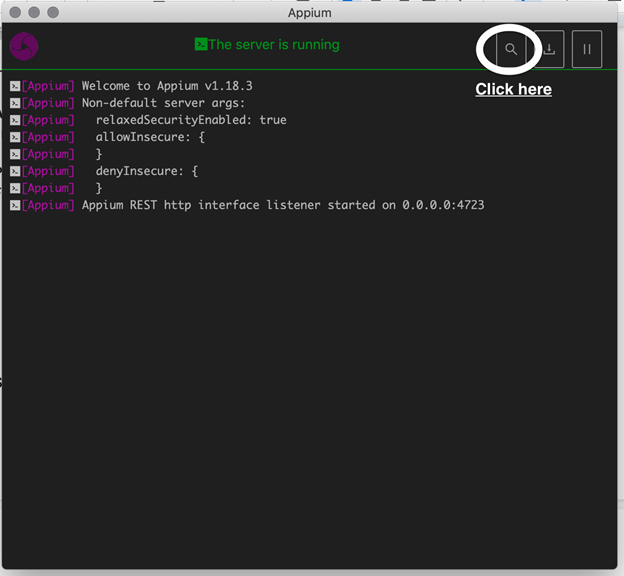
**Pre-requisites:**

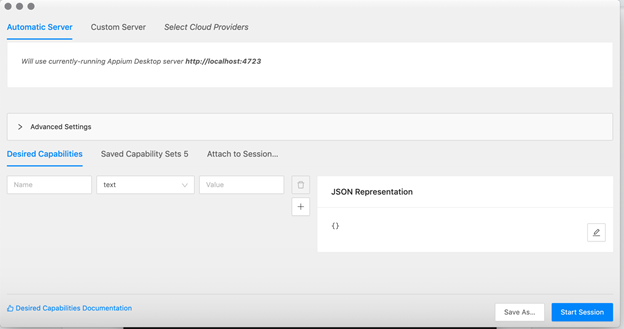
**For Android.**

* A standalone version of Appium (not via npm) is installed.
* The device/emulator is up and running.
* USB debugging is enabled.
* Device is detected by adb.
* The apk is available.
* The apk is not available and the app is already installed then we can use app package and activity.

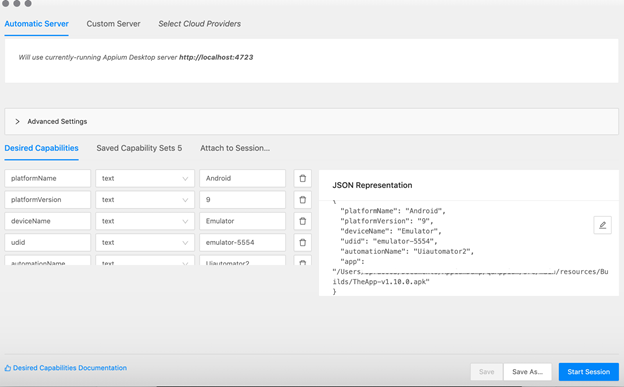
**Steps:**

1. Launch Appium server(older version) or appium inspector
2. This will show the below window.

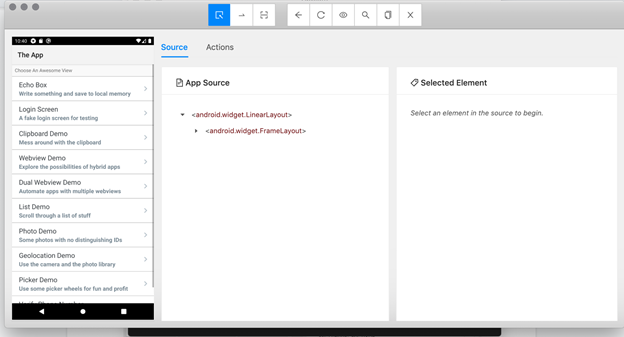
****       3. Click to the search button, this will take you to the next setting desired capabilities screen.

****

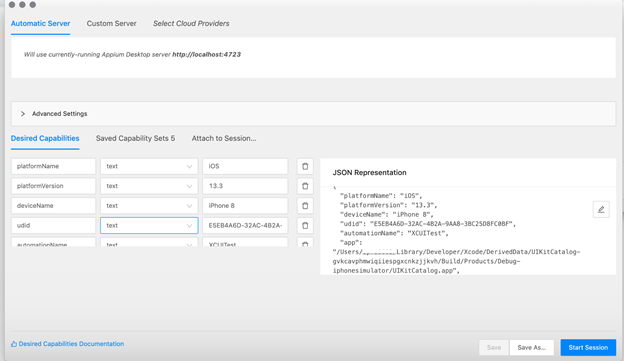
     4. Set the correct desired capabilities as your device/emulator/simulator and click to start the session.

****

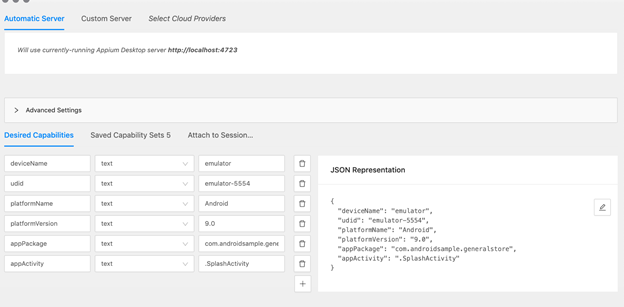
      5. And we have reached the screen from which we can inspect the element.

****

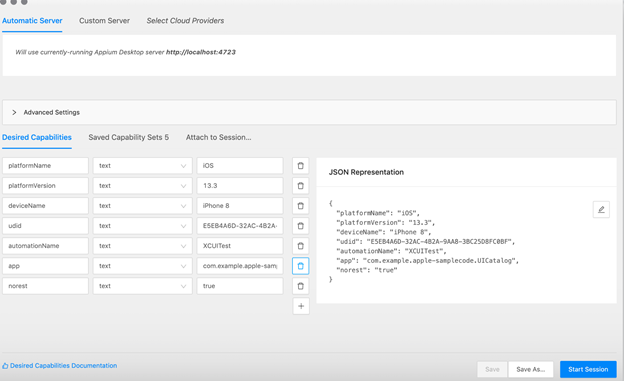
       6. For iOS, we can set the desired capabilities as

****

     7. If app is already installed, we can use as below (Android)

****

     8. For iOS

****

**What is uiautomator & uiautomation?**

**Uiautomator:** Appium uses Uiautomator to execute commands on real devices (Android) and emulators. Uiautomator is Google's test framework for mobile app automation at the UI level.

**Uiautomation:** Appium uses Uiautomation to execute commands on real devices (IOS) and simulators. Uiautomation is ios test framework for mobile app automation at the UI level.

**UIAutomator Viewer:**

Graphical user interface, text, application

Description automatically generated

* **Setting up UIAutomatorViewer**

**Prerequisites for setting up UIAutomatorViewer**

1. Install Appium, Install Android Studio and SDK Tools
2. Install Java and set up the environment variables
3. Install Eclipse IDE for Java Connect the Android device where the test application is to be tested.
4. The system must have Appium Setup and should be up & running

* **Opening UIAutomator Viewer through SDK tools kit**

This can be done by either of the following methods:

1. entering uiautomatorviewer in the command prompt
2. opening uiautomatorviewer.bat file in the Android installation folder with the following navigation: Android >> Android-SDK >> Tools >> UIAutomatorViewer.bat

* **Inspecting Page elements using UIAutomatorviewer**

**Pre-requisites**:

1. The device/emulator is up and running.
2. USB debugging is enabled.
3. Device is detected by adb.
4. The desired app is opened on mobile.

**Steps:**

1. Go to the ~/ANDROID\_HOME/tools/bin/ folder.
2. Run the uiautomatorviewer batch file (for windows) or sh file (for mac).
3. This will open the windows as shown below.

Graphical user interface, application, Word

Description automatically generated

1. Then it will capture the screenshot and show it with Page object hierarchy and attributes with values.

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Note: UIAutomatorviewer works with Android native app elements only.**

* **Understanding different locator strategies**

1. ID
2. Class Name.
3. Xpath
4. Accessibility ID
5. Android UIAutomator
6. Android View Tag (Espresso Only)
7. iOS UI
8. Automation

* **Taking the Device screenshot**

This option used to take device screenshot with complete hierarchy

* **Device screenshot with Compressed hierarchy**

This option used to take device screenshot with compressed hierarchy

* **NAF toggle**

NAF stands for Not Accessibility Friendly

We're looking for UI controls that are enabled, clickable but have no **text** nor **content-description**. Such controls configuration indicates an **interactive control** is present in the UI and is most likely not **accessibility** friendly. We refer to such controls here as NAF controls.

NAF doesn’t stop you from automating them by using locators. You can still use **XPATHs** to query the child element from a unique parent/ancestor element, although it might be a longer XPATH expression than simply pointing straight to a node that is correctly labelled with an ID.

**Check Android SDK Versions**:

**command line**: avdmanager list target

or

**command line**: android list target

or

**command line**: adb -s emulator-5554 shell getprop ro.build.version.release

**DesiredCapabilities for Android & IOS**:

Ref: <https://appium.io/docs/en/writing-running-appium/caps/index.html>

**Install APK in Virtual Device & Real Device using Appium.**

Step1: Start emulator or connect real device with USB or Wifi

Step2. Launch Appium and start server

Step3: Launch Appium inspector

Step4: Under desired Capabilities tab add respective capabilities like platformName, devicename, app, platformversion etc.

Step5: Click on start session.

Example:

{

"appium:automationName": "Appium",

"platformName": "Android",

"appium:platformVersion": "13",

"appium:deviceName": "emulator-5554",

"appium:app": "C:\\Users\\spittala\\Downloads\\ApiDemos-debug.apk"

}

Graphical user interface, application

Description automatically generated

**Install APK in Virtual Device & Real Device using adb.exe**

Step1: Start emulator or connect real device with USB or Wi-Fi.

Step2. Launch Appium and start server

Step3: open command prompt and execute below line

adb install <apk path>

**Install APK in Virtual Device & Real Device by script.**

**Script:**

DesiredCapabilities ds = **new** DesiredCapabilities();

ds.setCapability(MobileCapabilityType.***AUTOMATION\_NAME***, "Appium");

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

ds.setCapability(MobileCapabilityType.***PLATFORM\_VERSION***, "7.1");

ds.setCapability(MobileCapabilityType.***DEVICE\_NAME***, "emulator -5554"); //For emulator

//or

// ds.setCapability(MobileCapabilityType.***DEVICE\_NAME***, "Android"); // Real Device

// ds.setCapability(MobileCapabilityType.APP, "D:\\Selenium\_SS\\ ApiDemos-debug.apk");

or

ds.setCapability(MobileCapabilityType.APP, "https://github.com/appium/appium/raw/master/sample-code/apps/ApiDemos-debug.apk");

ds.setCapability("appPackage", "com.finance.emi.calculate");

ds.setCapability("appActivity", "com.finance.emi.calculate.ui.SplashActivity");

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

AndroidDriver<WebElement> driver = **new** AndroidDriver<WebElement>(url,ds);

driver.manage().timeouts().implicitlyWait(20, TimeUnit.***SECONDS***);

Thread.*sleep*(10000);

driver.findElementById("com.finance.emi.calculate:id/btn\_skip").click();

driver.findElementById("com.finance.emi.calculate:id/rootView").click();

driver.findElementByXPath("//android.widget.EditText[@resource-id='com.finance.emi.calculate:id/principal']").sendKeys("1000000");

driver.findElementById("com.finance.emi.calculate:id/interest").sendKeys("10.25");

driver.findElementById("com.finance.emi.calculate:id/loan\_tenure").sendKeys("5");

driver.findElementByXPath("//android.widget.Button[@text='Calculate']").click();

Thread.*sleep*(10000);

driver.quit();

**Different types of locators in Appium?**

Ref: <https://appium.io/docs/en/commands/element/find-elements/index.html#selector-strategies>

|  |  |
| --- | --- |
| **Locator** | **attributes selection** |
| ID | android : resource-id & IOS : name |
| name | Name |
| Class name | Class name |
| Accessibility ID | android : content-desc & IOS : accessibility-id |
| xpath | xpath |
| Android UiAutomator | [Use the UIAutomator API, in particular the UiSelector & UiScrollable class to locate elements. In Appium you send the Java code, as a string, to the server, which executes it in the application’s environment, returning the element or elements.](https://developer.android.com/reference/android/support/test/uiautomator/UiSelector.html) |
| Android View Tag | Locate an element by its view tag |
| IOS UiAutomation | [Use the UIAutomator API, in particular the UiSelector & UiScrollable class to locate elements. In Appium you send the Java code, as a string, to the server, which executes it in the application’s environment, returning the element or elements.](https://developer.android.com/reference/android/support/test/uiautomator/UiSelector.html) |
| Image | Locate an element by matching it with a base 64 encoded image file |

1. **ID**

Each element has a unique ID which makes easier to identify.

**resource-id** is used as an element identifier for Android and **name** is used for iOS.

**Example:**

driver.findElementById("IntegerA"); // for **iOS**

driver.findElementById("android:id/text1")).click(); //for **Android**

1. **Accessibility ID**

It is a highly preferred locator strategy, especially in the case of automating Android and iOS test cases. Developers can explicitly set the Accessibility ID during development.

For iOS, the default **Accessibility ID** is set to the **name** of the UI element. For Android, the value of Accessibility is same as the value of the attribute “**content-desc**”.

**Example:**

driver.findElementByAccessibilityId("Accessibility").click();

1. **ClassName**

Finding an element using class name is very common, but multiple elements may have the same class name, and this creates a problem in finding one particular element.

So, we need to use a combination of multiple attributes, for example, combining text with the class name to identify the element.

For iOS, Class Name is represented as the full name of the XCUI element and begins with XCUIElementType.

**For example** – UIAButton, UIARadioButton

In the case of Android, the Class Name is called out as the full name of the UIAutomator2 class. **For example** – android.widget.TextView

**Example** :

List<WebElement> buttons = driver.findElementsByClassName("android.widget.TextView");

for(WebElement button : buttons)

{

System.out.println(button.getText());

if(button.getText().equals("Animation"))

{

button.click();

}

}

1. **Xpath**

We should go xpath only when there is no ID, Name, or accessibility ID assigned to a specific UI element.



**Example:**

driver.findElementByXPath("(//XCUIElementTypeButton)[1]");

driver.findElementByXPath("//android.widget.Button[@text='Calculate']").click();

driver.findElementByXPath("//android.widget.EditText[@resource-id='com.finance.emi.calculateid/principal']").click();

1. **Android UI Automator**

This locator is Android-specific. we need to use the UI Automator API, i.e. **UISelector** Class to search for specific elements.

**Ref** : [https://developer.android.com/training/testing/ui-automator#java](https://developer.android.com/training/testing/ui-automator%23java)

**Example:**

String selector = "new UiSelector().text(“Cancel”)).className(“android.widget.Button”))"; MobileElement element = (MobileElement)driver.findElement(MobileBy.AndroidUIAutomator(selector));

**UiSelector class** : Represents a query for one or more target UI elements on a device.

**methods:**

1. text()

**Example:**

driver.findElementByAndroidUIAutomator("new UiSelector().text(\"Review Tabs\")").click();

1. textContains(“xxxx”)

**Example:**

driver.findElementByAndroidUIAutomator("new UiSelector().textContains(\"Tabs\")").click();

1. textStartsWith(“xxxx”)

**Example:**

driver.findElementByAndroidUIAutomator("new UiSelector().textStartsWith(\"Review\")").click();

1. textMatches(“.\*app”)

**Example:**

driver.findElementByAndroidUIAutomator("new UiSelector().textMatches(\".\*view\")").click();

1. description(“xxxx”)

**Example:**

driver.findElementByAndroidUIAutomator("new UiSelector().description(\"android.widget.Btn\"))").click();

1. descriptionContains(“xxxx”)

**Example:**

driver.findElementByAndroidUIAutomator("new UiSelector().descriptionContains(\"widget.Btn\"))").click();

1. descriptionStartsWith(“xxxx”)

**Example:**

driver.findElementByAndroidUIAutomator("new UiSelector().descriptionStartsWith(\"android.widget \"))").click();

1. descriptionMatches(“.\*app”)

**Example:**

driver.findElementByAndroidUIAutomator("new UiSelector().descriptionMatches(\".\*widget.btn\"))").click();

1. resourceId(“xxxx”)

**Example:**

driver.findElementByAndroidUIAutomator("new UiSelector().resourceId(\"com.finance.emi.calculate:id/rootView\").index(1)").click();

1. resourceIdMatches(“.\*app”)

**Example:**

driver.findElementByAndroidUIAutomator("new UiSelector().resourceIdMatches(\".\*calculate:id/rootView\").index(1)").click();

1. className(“xxxx”)

**Example:**

driver.findElementByAndroidUIAutomator("new UiSelector().className(\"android.widget.EditText\").resourceId(\"com.finance.emi.calculate:id/principal\")").sendKeys("10");

1. classNameMatches(“.\*app”)

**Example:**

driver.findElementByAndroidUIAutomator("new UiSelector().classNameMatches(\".\*EditText\").resourceId(\"com.finance.emi.calculate:id/principal\")").sendKeys("10");

1. instance(index)

**Example:**

driver.findElementByAndroidUIAutomator("new UiSelector().className(\"android.widget.EditText\").instance(1)").click();

1. index(x)3

**Example:**

driver.findElementByAndroidUIAutomator("new UiSelector().className(\"android.widget.EditText\").index(1)").click();

1. clickable(true)

**Example:**

driver.findElementByAndroidUIAutomator("new UiSelector().resourceId(\"com.finance.emi.calculate:id/extraRepaymentCard\").clickable(true)").click();

1. scrollable(true)) :

**Example:**

driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector().scrollable(true)).scrollIntoView(new UiSelector().className(\"android.widget.Button\").index(2))").click();

**scrolling in Appium :**

**UiScrolable** class : Provides support for searching for items in a scrollable UI container.

**methods:**

1. flingForward(performs quick swipe)

**Example** :

**try** {

driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new UiSelector().scrollable(true)).flingForward()"));

} **catch**(InvalidSelectorException e) {

}

1. flingBackward(performs quick swipe)

Example :

**try**

{

driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new UiSelector().scrollable(true)).flingBackward()"));

}

**catch**(InvalidSelectorException e) {

}

1. scrollForward(moves exactly one view)

**Example** :

**try** {

driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new UiSelector().scrollable(true)).scrollForward()"));

} **catch**(InvalidSelectorException e) {

}

1. scrollBackward(moves exactly one view)

**Example** :

**try** {

driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new UiSelector().scrollable(true)).scrollBackward()"));

}**catch** (InvalidSelectorException e) {

}

1. scrollToBeginning(10) moves exactly by one view. 10 scrolls max)

**Example** :

**try** {

driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new UiSelector().scrollable(true)).scrollToBeginning(10)"));

} **catch** (InvalidSelectorException e) {

}

1. flingToBeginning(10) performs quick swipes. 10 swipes max)

**Example** :

**try** {

driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new UiSelector().scrollable(true)).flingToBeginning(10)"));

} **catch** (InvalidSelectorException e) {

}

1. scrollToEnd(10) moves exactly by one view. 10 scrolls max)

**Example** :

**try** {

driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new UiSelector().scrollable(true)).scrollToEnd(10)"));

} **catch** (InvalidSelectorException e) {

}

1. flingToEnd(10) performs quick swipes. 10 swipes max)

**Example :**

**try** {

driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new UiSelector().scrollable(true)).flingToEnd(10)"));

} **catch** (InvalidSelectorException e) {

}

1. scrollIntoView(new UiSelector().className(“xxxx”))

**Example**:

driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector()).scrollIntoView(new UiSelector().description(\"android.widget.Buttn\"))").click();

or

driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector().resourceId(\"android:id/list\")).scrollIntoView(new UiSelector().text(\"Share Result\").instance(0)").click();

1. getChildByText(new UiSelector().className(“xxxx”),”text”)

**Example**:

driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector().resourceId(\"android:id/list\")).getChildByText(new UiSelector().className(\"android.widget.Button\"),\"Share Result\")").click();

1. getChildByDescription(new UiSelector().className(“xxxx”),”description”)

**Example**:

driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector().resourceId(\"android:id/list\")).getChildByDescription(new UiSelector().className(\"android.widget.Button\"),\"Share Result\")").click();

1. setAsHorizontalList()

**Example**:

driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector().resourceId(\"android:id/list\")).setAsHorizontalList().scrollIntoView(new UiSelector().text(\"Share Result\").instance(0)").click();

or

driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector().resourceId(\"android:id/list\")).setAsHorizontalList().getChildByText(new UiSelector().className(\"android.widget.Button\"),\"Share Result\")").click();

1. **Android View Tag**

This is also an Android platform-specific locator. We use this to locate elements using its view tag.

**Example:**

driver.findElementByAndroidViewTag("android.widget.Button").click();

**AndroidDriver class methods:**

1. launchApp(): This method used to launch application.

**Example**:

driver.launchApp();

1. installApp(String appPath) : This method used to Install the given app onto the device.

**Example**:

driver.installApp("D:\\xxxx\\xxx.apk");

1. isAppInstalled(java.lang.String bundleId); This method used to Check whether the specified app is installed on the device. it returns Boolean value.

Note : Need to pass app package as parameter.

**Example:**

**boolean** status = driver.isAppInstalled("io.appium.android.apis");

System.***out***.println(status);

1. resetApp() : This method used Reset the currently running app for this session

**Example:**

driver.resetApp();

1. runAppInBackground(Duration); This method Send the currently running app for this session to the background

**Example**:

driver.runAppInBackground(Duration.*ofSeconds*(10));

1. removeApp(java.lang.String bundleId); This method used Remove an app from the device.

**Example:**

driver.removeApp("com.example.AppName");

1. closeApp(): Close an app on device

**Example**:

driver.closeApp();

1. activateApp(String bundleId); Activate the given app onto the device

**Example:**

driver.activateApp('com.apple.Preferences');

driver.activateApp('io.appium.android.apis');

1. terminateApp(String bundleId) : Terminate the given app on the device.

**Example:**

driver.terminateApp(“com.apple.Preferences”);

driver.terminateApp(“io.appium.android.apis”);

1. getPlatformName() : Get the name of the current platform.

**Example**:

String **platform** = driver. getPlatformName();

1. getCurrentPackage() : Get the name of the current Android package

**Example**:

String **package** = driver.getCurrentPackage();

1. currentActivity() : Get the name of the current Android activity

**Example**:

String activity = driver.currentActivity();

1. getDeviceTime() : Get the time on the device

**Example**:

String time = driver.getDeviceTime();

1. getScreenshotAs() : Take a screenshot of the current viewport/window/page.

**Example**:

File scrFile = driver.getScreenshotAs(OutputType.FILE);

1. rotate(ScreenOrientation):Rotate the device in 3 dimentions.

**Example**:

driver.rotate(org.openqa.selenium.ScreenOrientation.***PORTRAIT***);

1. isBrowser() : Check whether opened app is browser or not

**Example**:

boolean browser = driver.isBrower();

1. isDeviceLocked(): Check whether the device is locked or not

**Example**:

**boolean** isLocked = driver.isDeviceLocked();

1. isKeyboardShown() : Check whether keyboard as displayed or not

**Example**:

**boolean** keyboard = driver.isKeyboardShown();

1. lockDevice(): Lock the device

**Example**:

driver.lockDevice();

1. lockDevice(Duration) : Lock the device for given time.

**Example**:

driver.lockDevice(Duration.ofSeconds(10));

1. unlockDevice() :Unlock the device.

**Example**:

driver.unlockDevice();

1. openNotifications() : open the notifications

**Example**:

driver.openNotifications();

1. getKeyboard() : display keyboard.

**Example**:

driver.getKeyboard();

1. hideKeyboard() : Hide soft keyboard

**Example**:

driver.hideKeyboard();

1. isKeyboardShown() : Whether or not the soft keyboard is shown

**Example**:

boolean isKeyboardShown = driver.isKeyboardShown();

1. longPressKey(AndroidKeyCode) : Press and hold a particular key code on the device

Example:

driver.longPressKey(**new** KeyEvent(AndroidKey.***HOME***));

1. pressKey(AndroidKeyCode) : Press a particular key code on the device

Example:

driver.pressKey(**new** KeyEvent(AndroidKey.***HOME***));

1. getContext() : Get the current context (application type) in which Appium is running

**Example**:

String context = driver.getContext();

1. getContextHandles() : Get all contexts (application type) in which Appium is running

**Example**:

Set<String> contexts = driver.getContextHandles();

1. Context() : switch to context (application type) which is passed as parameter.

**Example**:

driver.Context(“CHROMIUM”);

1. setClipboardText(string text) : Set the content of the system clipboard

**Example**:

driver.setClipboardText("happy testing");

1. getClipboardText(string text) : Get the content of the system clipboard

Example:

String clipboardtext = driver.getClipboardText();

1. setPowerAC(PowerACState.***ON***) : For Android emulator. To set the state of the battery charger to connected or not.

**Example**:

driver.setPowerAC(PowerACState.***ON***);

1. setPowerCapacity(int) : For Android emulator. To set the battery percentage.

**Example**:

driver.setPowerCapacity(100);

1. pullFile(path) : Retrieve a file from the device's file system

**Example:**

**byte**[] cfile = driver.pullFile("/storage/emulated/0/Download/androidss.png");

**try** ( FileOutputStream outputStream = **new** FileOutputStream("D:\\Selenium\_SS\\android012.png");)

{outputStream.write(cfile);}

**catch** (Exception e)

{e.printStackTrace();}

1. pushFile(path) : Retrieve a file from the device's file system

**Example:**

File file = **new** File("D:\\Selenium\_SS\\android012.png");

driver.pushFile("/storage/emulated/0/Download/and01.png", file);

1. makeGsmCall() : Make call (Emulator only)

Example:

driver.makeGsmCall("555-123-4567", GsmCallActions.CALL);

1. sendSMS() : Send sms message (Emulator only)

Example:

driver.sendSMS("555-123-4567",”Hellow”);

1. toggleAirplaneMode() :Toggle airplane mode on device

Example:

driver.toggleAirplaneMode();

1. toggleData() : Switch the state of data service

Example:

driver.toggleData();

Note : (For Android) This API does not work for Android API level 21+ because it requires system or carrier privileged permission, and Android <= 21 does not support granting permissions.

1. toggleLocationServices() : Switch the state of the location service

Example:

driver.toggleLocationServices();

1. toggleWifi() : Switch the state of the wifi service

Example:driver.toggleWifi();

1. getPerformanceData() : Returns the information of the system state, which is supported to read as like cpu, memory, network traffic, and battery.

Example:

List<List<Object>> performanceData = driver.getPerformanceData("my.app.package","cpuinfo", 5);

1. quit() : End the running session

Example:

driver.quit();

1. getSessionDetails() : Retrieve the capabilities of the specified session

Example:

Map<String, Object> caps = driver.getSessionDetails();

1. location() :Get the current geo location

Example :

Location location = driver.location();

1. back() : Navigate back

Example :

driver.navigate().back();

1. refresh() : refresh the page

Example :

driver.navigate().refresh();

1. forword() : Navigate forword

Example :

driver.navigate().forword();

**How to switch native view to WebView in appium:**

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

Set<String> contextNames = driver.getContextHandles();

for (String contextName : contextNames)

{

System.out.println(contextNames); //prints out something like NATIVE\_APP \n WEBVIEW\_1

}

driver.context(contextNames.toArray()[1]); // set context to WEBVIEW\_1

**or**

driver.context("WEBVIEW\_1");

//do some web testing

String myText = driver.findElement(By.cssSelector(".green\_button")).click();

driver.context("NATIVE\_APP");

// do more native testing if we want

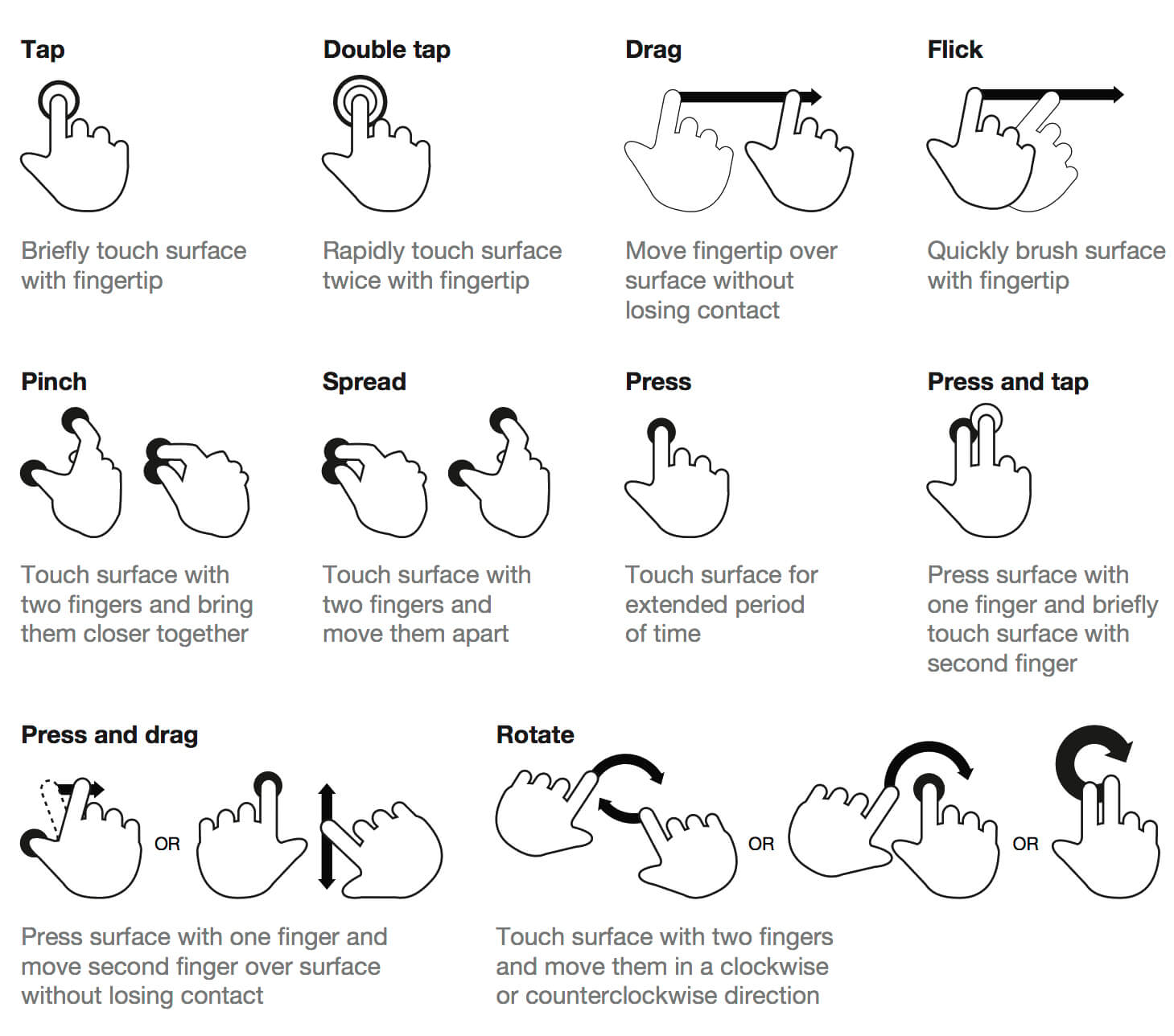
driver.quit();

**Mobile Gestures:**

Gesture control apps allow you to control your smartphone by way of simple hand movements or swipe gestures. These super helpful apps are all about making it easier to interact with your device

These are some of the most common user gestures:

* **Tap**: Touch surface briefly
* **Double tap**: Touch surface with two quick motions (often to zoom)
* **Drag**: Move along surface without breaking contact
* **Pinch/spread**: Touch surface with two fingers to move in (pinch) or out (spread)
* **Press**: Touch surface and hold
* **Flick**: Scrolls quickly



**Appium supports the following gestures:**

* Tap on an element.
* Tap on x, y coordinates.
* Press an element for a particular duration.
* Press x, y coordinates for a particular duration.
* Horizontal swipe: Using start and end percentage of the screen height and width.
* Vertical swipe: Using start and end percentage of the screen height and width.
* Drag (Swipe) one element to another element.
* Multitouch for an element.

Appium supports these gestures using the TouchAction class & JavascriptExecutor

**how can you get the x, y coordinate?**

We can know the coordinates by 3 ways

1. Pointer location in Android & IOS
   1. Move to Settings > Developer options
   2. Enable the Pointer location.
   3. Now move to any application for which you need the coordinates of a particular location. Tap on the location and you will get the coordinations for that place at top of the screen.

Graphical user interface

Description automatically generated Table

Description automatically generated

Figure-1: Enable the Pointer location.

1. Appium Inspector (swipe by co-ordinates and tap by co-ordinates)
2. Appium getSize() method

**TouchAction Class in appium:**

Touchaction class used to perform interactive action on mobile application.

1. longPress(PointOption) : Perform long press on element.

**Syntax**:  
TouchAction action = **new** TouchAction(driver);

action.longPress(element).perform();

**Example**:

action.longPress(ElementOption.*element*(source)).moveTo(ElementOption.*element*(dest)).release().perform();

1. tap() : Perform single tap on element.

**Example**:

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

action.tap(element).perform();

or

TouchActions action = **new** TouchActions(driver);

action.tap(PointOption.*point*(115, 1083)).perform();

1. doubleTap(): Perform double tap on element.

Example:

TouchAction action = new TouchAction(driver);

action.doubleTap(element).perform();

1. singleTap(): Perform single tap on element.

Example:

TouchAction action = new TouchAction(driver);

action.singleTap(element).perform();

1. waitAction(waitOptions): Perform wait between the actions.

**Example**:

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

action.tap(PointOption.*point*(115,1083)).waitAction(

WaitOptions.waitOptions(Duration.ofSeconds(0))).perform();

1. press(PointOption) : Preform press action on element.

Example:

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

action.press(element).perform();

or

TouchActions action = **new** TouchActions(driver);

action.press(PointOption.*point*(115,1083)).moveTo(PointOption.*point*(115, 1083)).release().perform();

1. release() : Preform release action on element.

Example:

TouchActions action = **new** TouchActions(driver);

action.release(element).perform();

1. moveTo(PointOption) : Perform moveTo an element.

Example:

TouchActions action = **new** TouchActions(driver);

action.press(PointOption.*point*(115,1083)).moveTo(PointOption.*point*(115, 1083)).release().perform();

**MultiTouchAction:**

* MultiTouch objects are collections of TouchActions.
* MultiTouch gestures only have two methods, add, and perform.
* add is used to add another TouchAction to this MultiTouch.
* When perform is called, all the TouchActions which were added to the MultiTouch are sent to appium and performed as if they happened at the same time. Appium first performs the first event of all TouchActions together, then the second, etc.

**Example of tapping with two fingers:**

action0 = **new** TouchAction(driver).tap(ele1);

action1 = **new** TouchAction(driver).tap(ele2);

**new** MultiTouchAction(driver).add(action0).add(action1).perform();

**Automating Mobile Gestures with UiAutomator2 Backend:**

1. **mobile: longClickGesture**

This gesture performs long click action on the given element/coordinates. Available since Appium v1.19

**Supported arguments**

* elementId: The id of the element to be clicked. If the element is missing then both click offset coordinates must be provided. If both the element id and offset are provided, then the coordinates are parsed as relative offsets from the top left corner of the element.
* x: The x-offset coordinate
* y: The y-offset coordinate
* duration: Click duration in milliseconds. 500 by default. The value must not be negative

**Example:**

((JavascriptExecutor) driver).executeScript("mobile: longClickGesture", ImmutableMap.of("elementId", ((RemoteWebElement) element).getId()));

or

JavascriptExecutor js = (JavascriptExecutor) driver;

HashMap<String, String> scrollObject = new HashMap<String, String>();

scrollObject.put("elementId ", ((RemoteWebElement) element).getId());

js.executeScript("mobile: longClickGesture", scrollObject);

1. **mobile: doubleClickGesture**

This gesture performs double click action on the given element/coordinates. Available since Appium v1.21

**Supported arguments**

* elementId: The id of the element to be clicked. If the element is missing then both click offset coordinates must be provided. If both the element id and offset are provided, then the coordinates are parsed as relative offsets from the top left corner of the element.
* x: The x-offset coordinate
* y: The y-offset coordinate

**Example:**

((JavascriptExecutor) driver).executeScript("mobile: doubleClickGesture", ImmutableMap.of( "elementId", ((RemoteWebElement) element).getId()));

or

JavascriptExecutor js = (JavascriptExecutor) driver;

HashMap<String, String> scrollObject = new HashMap<String, String>();

scrollObject.put("elementId ", ((RemoteWebElement) element).getId());

js.executeScript("mobile: doubleClickGesture", scrollObject);

1. **mobile: clickGesture**

This gesture performs click action on the given element/coordinates. Available since Appium UiAutomator2 driver 1.71.0. Usage of this gesture is recommended as a possible workaround for cases where the "native" tap call fails, even though tap coordinates seem correct. This issue is related to the fact these calls use the legacy UIAutomator-based calls while this extension is based on the same foundation as W3C does.

**Supported arguments**

* elementId: The id of the element to be clicked. If the element is missing, then both click offset coordinates must be provided. If both the element id and offset are provided, then the coordinates are parsed as relative offsets from the top left corner of the element.
* x: The x-offset coordinate
* y: The y-offset coordinate

**Example**:

((JavascriptExecutor) driver)..executeScript("mobile: clickGesture", ImmutableMap.of("elementId", ((RemoteWebElement) element).getId()));

or

JavascriptExecutor js = (JavascriptExecutor) driver;

HashMap<String, String> scrollObject = new HashMap<String, String>();

scrollObject.put("elementId ", ((RemoteWebElement) element).getId());

js.executeScript("mobile: clickGesture", scrollObject);

1. **mobile: dragGesture**

This gesture performs drag action from the given element/coordinates to the given point. Available since Appium v1.19

**Supported arguments**

* elementId: The id of the element to be dragged. If the element id is missing then both start coordinates must be provided. If both the element id and the start coordinates are provided then these coordinates are considered as offsets from the top left element corner.
* startX: The x-start coordinate
* startY: The y-start coordinate
* endX: The x-end coordinate. Mandatory argument
* endY: The y-end coordinate. Mandatory argument
* speed: The speed at which to perform this gesture in pixels per second. The value must not be negative. The default value is 2500 \* displayDensity

**Example:**

((JavascriptExecutor) driver).executeScript("mobile: dragGesture", ImmutableMap.of( "elementId", ((RemoteWebElement) element).getId(),

"endX", 100,

"endY", 100));

Or

((JavascriptExecutor) driver).executeScript("mobile: dragGesture", ImmutableMap.of( "startX", 250, "startY", 250,

"endX", 100,

"endY", 100));

Or

JavascriptExecutor js = (JavascriptExecutor) driver;

HashMap<String, String> scrollObject = new HashMap<String, String>();

scrollObject.put("elementId ", ((RemoteWebElement) element).getId());

js.executeScript("mobile: dragGesture", scrollObject);

Or

JavascriptExecutor js = (JavascriptExecutor) driver;

HashMap<String, String> scrollObject = new HashMap<String, String>();

scrollObject.put"startX", 100);

scrollObject.put"startY", 100);

scrollObject.put"endX", 250);

scrollObject.put"endY", 250);

js.executeScript("mobile: dragGesture", scrollObject);

1. **mobile: flingGesture**

* This gesture performs fling gesture on the given element/area. Available since Appium v1.19

**Supported arguments**

* elementId: The id of the element to be flinged. If the element id is missing, then fling bounding area must be provided. If both the element id and the fling bounding area are provided, then this area is effectively ignored.
* left: The left coordinate of the fling bounding area
* top: The top coordinate of the fling bounding area
* width: The width of the fling bounding area
* height: The height of the fling bounding area
* direction: Direction of the fling. Mandatory value. Acceptable values are: up, down, left and right (case insensitive)
* speed: The speed at which to perform this gesture in pixels per second. The value must be greater than the minimum fling velocity for the given view (50 by default). The default value is 7500 \* displayDensity
* Returned value
* The returned value is a boolean one and equals to true if the object can still scroll in the given direction

**Examples:**

boolean canScrollMore = (Boolean) ((JavascriptExecutor) driver).executeScript("mobile: flingGesture", ImmutableMap.of(

"elementId", ((RemoteWebElement) element).getId(),

"direction", "down",

"speed", 500));

Or

JavascriptExecutor js = (JavascriptExecutor) driver;

HashMap<String, String> scrollObject = new HashMap<String, String>();

scrollObject.put"left", 100);

scrollObject.put"top", 100);

scrollObject.put"width", 250);

scrollObject.put"height", 250);

scrollObject.put" direction ", “down”);

scrollObject.put" speed ", 500);

boolean canScrollMore = (Boolean) js.executeScript("mobile: flingGesture",scrollObject);

1. **mobile: pinchOpenGesture**

* This gesture performs pinch-open gesture on the given element/area. Available since Appium v1.19

**Supported arguments:**

* elementId: The id of the element to be pinched. If the element id is missing, then pinch bounding area must be provided. If both the element id and the pinch bounding area are provided, then the area is effectively ignored.
* left: The left coordinate of the pinch bounding area
* top: The top coordinate of the pinch bounding area
* width: The width of the pinch bounding area
* height: The height of the pinch bounding area
* percent: The size of the pinch as a percentage of the pinch area size. Valid values must be float numbers in range 0..1, where 1.0 is 100%. Mandatory value.
* speed: The speed at which to perform this gesture in pixels per second. The value must not be negative. The default value is 2500 \* displayDensity

**Examples:**

((JavascriptExecutor) driver).executeScript("mobile: pinchOpenGesture", ImmutableMap.of("elementId", ((RemoteWebElement) element).getId(),

"percent", 0.75));

Or

JavascriptExecutor js = (JavascriptExecutor) driver;

HashMap<String, String> scrollObject = new HashMap<String, String>();

scrollObject.put"left", 100);

scrollObject.put"top", 100);

scrollObject.put"width", 250);

scrollObject.put"height", 250);

scrollObject.put" percent", 0.5);

scrollObject.put" speed ", 500);

js.executeScript("mobile: pinchOpenGesture",scrollObject);

1. **mobile: pinchCloseGesture**

* This gesture performs pinch-close gesture on the given element/area. Available since Appium v1.19

**Supported arguments**

* + elementId: The id of the element to be pinched. If the element id is missing then pinch bounding area must be provided. If both the element id and the pinch bounding area are provided then the area is effectively ignored.
  + left: The left coordinate of the pinch bounding area
  + top: The top coordinate of the pinch bounding area
  + width: The width of the pinch bounding area
  + height: The height of the pinch bounding area
  + percent: The size of the pinch as a percentage of the pinch area size. Valid values must be float numbers in range 0..1, where 1.0 is 100%. Mandatory value.
  + speed: The speed at which to perform this gesture in pixels per second. The value must not be negative. The default value is 2500 \* displayDensity

**Examples:**

((JavascriptExecutor) driver).executeScript("mobile: pinchCloseGesture", ImmutableMap.of(

"elementId", ((RemoteWebElement) element).getId(),

"percent", 0.75));

Or

JavascriptExecutor js = (JavascriptExecutor) driver;

HashMap<String, String> scrollObject = new HashMap<String, String>();

scrollObject.put"left", 100);

scrollObject.put"top", 100);

scrollObject.put"width", 250);

scrollObject.put"height", 250);

scrollObject.put" percent", 0.5);

scrollObject.put" speed ", 500);

js.executeScript("mobile: pinchCloseGesture",scrollObject);

1. **mobile: swipeGesture**

* This gesture performs swipe gesture on the given element/area. Available since Appium v1.19

**Supported arguments**

* elementId: The id of the element to be swiped. If the element id is missing then swipe bounding area must be provided. If both the element id and the swipe bounding area are provided then the area is effectively ignored.
* left: The left coordinate of the swipe bounding area
* top: The top coordinate of the swipe bounding area
* width: The width of the swipe bounding area
* height: The height of the swipe bounding area
* direction: Swipe direction. Mandatory value. Acceptable values are: up, down, left and right (case insensitive)
* percent: The size of the swipe as a percentage of the swipe area size. Valid values must be float numbers in range 0..1, where 1.0 is 100%. Mandatory value.
* speed: The speed at which to perform this gesture in pixels per second. The value must not be negative. The default value is 5000 \* displayDensity

**Examples:**

((JavascriptExecutor) driver).executeScript("mobile: swipeGesture", ImmutableMap.of(

"left", 100, "top", 100, "width", 200, "height", 200,

"direction", "left",

"percent", 0.75));

Or

JavascriptExecutor js = (JavascriptExecutor) driver;

HashMap<String, String> scrollObject = new HashMap<String, String>();

scrollObject.put"left", 100);

scrollObject.put"top", 100);

scrollObject.put"width", 250);

scrollObject.put"height", 250);

scrollObject.put" direction ", “left”);

scrollObject.put" percent ", 0.75);

js.executeScript("mobile: swipeGesture",scrollObject);

1. **mobile: scrollGesture**

* This gesture performs scroll gesture on the given element/area. Available since Appium v1.19

**Supported arguments**

* elementId: The id of the element to be scrolled. If the element id is missing then scroll bounding area must be provided. If both the element id and the scroll bounding area are provided then this area is effectively ignored.
* left: The left coordinate of the scroll bounding area
* top: The top coordinate of the scroll bounding area
* width: The width of the scroll bounding area
* height: The height of the scroll bounding area
* direction: Scrolling direction. Mandatory value. Acceptable values are: up, down, left and right (case insensitive)
* percent: The size of the scroll as a percentage of the scrolling area size. Valid values must be float numbers greater than zero, where 1.0 is 100%. Mandatory value.
* speed: The speed at which to perform this gesture in pixels per second. The value must not be negative. The default value is 5000 \* displayDensity

**Returned value**

* The returned value is a boolean one and equals to true if the object can still scroll in the given direction

**Examples:**

boolean canScrollMore = (Boolean) ((JavascriptExecutor) driver).executeScript("mobile: scrollGesture", ImmutableMap.of(

"left", 100, "top", 100, "width", 200, "height", 200,

"direction", "down",

"percent", 3.0));

**Or**

JavascriptExecutor js = (JavascriptExecutor) driver;

HashMap<String, String> scrollObject = new HashMap<String, String>();

scrollObject.put"left", 100);

scrollObject.put"top", 100);

scrollObject.put"width", 250);

scrollObject.put"height", 250);

scrollObject.put" direction ", “down”);

scrollObject.put" percent ", 0.75);

js.executeScript("mobile: scrollGesture",scrollObject);

**Alerts in appium:**

We can handle alert by treating alert components as normal webelements.

**Textbox, checkbox, radio button, taggle button**

**Toast message**

* We can handle toast message as a normal webelement in appium.

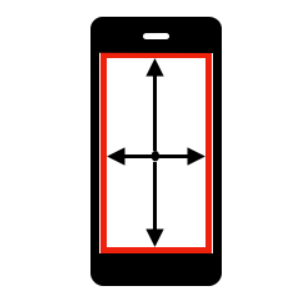
**Swipe in appium:**

We can perform swipe by using TouchAction or UiScrollable class in appium.

1. **Simple Screen swipe:**

**Ref:** [**http://appium.io/docs/en/writing-running-appium/tutorial/swipe/simple-screen/**](http://appium.io/docs/en/writing-running-appium/tutorial/swipe/simple-screen/)

* Swipe has start and end points. The start point of swipe is most important. The following elements may prevent swipe start: - application interface/menu e.g. header or footer - elements that are waiting tap and do not pass touch to scroll view.
* It is better to start swipe actions at the center of the screen to make them more reliable.



Example:

**public** **class** SwipeInAppium

{

**public** **void** swipeScreen(Direction dir)

{

System.***out***.println("swipeScreen(): dir: '" + dir + "'");

// always log your actions

// Animation default time:

// - Android: 300 ms

// - iOS: 200 ms

// final value depends on your app and could be greater

**final** **int** ANIMATION\_TIME = 200; // ms

**final** **int** PRESS\_TIME = 200; // ms

**int** edgeBorder = 10; // better avoid edges

PointOption pointOptionStart, pointOptionEnd;

// init screen variables

Dimension dims = driver.manage().window().getSize();

// init start point = center of screen

pointOptionStart = PointOption.point(dims.width / 2, dims.height / 2);

**switch** (dir)

{

**case** ***DOWN***: // center of footer

pointOptionEnd = PointOption.point(dims.width / 2, dims.height - edgeBorder);

**break**;

**case** ***UP***: // center of header

pointOptionEnd = PointOption.point(dims.width / 2, edgeBorder);

**break**;

**case** ***LEFT***: // center of left side

pointOptionEnd = PointOption.point(edgeBorder, dims.height / 2);

**break**;

**case** ***RIGHT***: // center of right side

pointOptionEnd = PointOption.point(dims.width - edgeBorder, dims.height / 2);

**break**;

**default**:

**throw** **new** IllegalArgumentException("swipeScreen(): dir: '" + dir + "' NOT supported");

}

// execute swipe using TouchAction

**try**

{

**new** TouchAction(driver)

.press(pointOptionStart)

// a bit more reliable when we add small wait

.waitAction(WaitOptions.waitOptions(Duration.ofMillis(PRESS\_TIME)))

.moveTo(pointOptionEnd)

.release().perform();

} **catch** (Exception e)

{

System.***err***.println("swipeScreen(): TouchAction FAILED\n" + e.getMessage());

**return**;

}

// always allow swipe action to complete

**try**

{

Thread.*sleep*(ANIMATION\_TIME);

}

**catch** (InterruptedException e)

{

// ignore

}

}

**public** **enum** Direction {

***UP***,

***DOWN***,

***LEFT***,

***RIGHT***;

}

}

**Date picker, time, and date change**.

**How to stop installing the App again and again?**

**a. Setting appium --no-reset Flag to true :** This Flag is used when we don't want to reset app state between sessions (Android: don’t uninstall app before new session). Its default Value is **False**,

We can start your Appium server with "appium --no-reset" command or include server capability like:   
 ds.setCapability("noReset", "true");

or

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

**b. Make Sure --full-reset Flag is False:** (Android) Reset app state by uninstalling app instead of clearing app data. On Android, this will also remove the app after the session is complete. Default **false**.  
 ds.setCapability("fullReset", "false");

or

ds.setCapability(MobileCapabilityType.***FULL\_RESET***, **false**);

**Run appium server through command line:**

To run appium server from command prompt we need to install below software’s

* Node.js :

Step 1: Download Node.js Installer from browser, navigate to https://nodejs.org/en/download/

Step 2: Install Node.js.

Step 3: Verify Installation by below command in cmd,

**npm -v**

**Note :** npm stands for Node Package Manager (software)

* Appium :

Step 1: Install appium by executing below command in cmd,

**npm install -g appium**

Step 2: Verify Installation by running appium with below command in cmd,

**appium -v**

Run appium execute the below command,

**appium &**

**Run appium server through script:**

* **Run appium with default service**

**Requrired jar files :**

1. **slf4j.jar**
2. **logback.jar**
3. **commons-validator.jar**

AppiumDriverLocalService appiumService = AppiumDriverLocalService.*buildDefaultService*();

appiumService.start();

DesiredCapabilities ds = **new** DesiredCapabilities();

ds.setCapability(MobileCapabilityType.***AUTOMATION\_NAME***, "Appium");

AndroidDriver<WebElement> driver = **new** AndroidDriver<WebElement>(appiumService.getUrl(),ds);

driver.manage().timeouts().implicitlyWait(20, TimeUnit.***SECONDS***);

driver.quit();

appiumService.stop();

* **Run appium with build service**

AppiumServiceBuilder asb = **new** AppiumServiceBuilder();

asb.withIPAddress("0.0.0.0");

asb.usingAnyFreePort();

// asb.usingPort(4723);

AppiumDriverLocalService appiumService = AppiumDriverLocalService.*buildService*(asb);

appiumService.start();

DesiredCapabilities ds = **new** DesiredCapabilities();

ds.setCapability(MobileCapabilityType.***AUTOMATION\_NAME***, "Appium");

AndroidDriver<WebElement> driver = **new** AndroidDriver<WebElement>(appiumService.getUrl(),ds);

driver.manage().timeouts().implicitlyWait(20, TimeUnit.***SECONDS***);

driver.quit();

appiumService.stop();

**How to setup iOS Automation testing on MacOS**

Step by Step

1. Java

2. Appium

3. Xcode (For simulators)

4. Xcode command line tools

5. WebDriverAgent

6. Real Devices OR Simulators

7. IDE (eclipse)

**Note**: We can install android studio and xcode in mac os, but we can not install xcode in window os.

So we can create emulator & simulator both in mac os but we can create only emulator on windows os.

**Sample program**:

import io.appium.java\_client.ios.IOSDriver;

import java.net.MalformedURLException;

import java.net.URL;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.openqa.selenium.remote.DesiredCapabilities;

public class Edition041\_iOS\_Real\_Device {

private IOSDriver driver;

@Before

public void setUp() throws MalformedURLException {

//Setting Desired Capabilities

DesiredCapabilities capabilities = new DesiredCapabilities();

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

capabilities.setCapability("platformVersion", "12.4.7");

capabilities.setCapability("deviceName", "iPhone 7");

capabilities.setCapability("udid", "<your iPhone’s udid>"); // Unique Device Identifier (UDID)

capabilities.setCapability("bundleId", "com.google.Chrome");

capabilities.setCapability("xcodeOrgId", "<your org id>");

capabilities.setCapability("xcodeSigningId", "<your signing id>");

capabilities.setCapability("updatedWDABundleId", "com.google.chrome.ios");

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

}

@After

public void tearDown() {

if (driver != null) {

driver.quit();

}

}

@Test

public void testFindingAnElement() {

driver.findElementByAccessibilityId("Login Screen");

}

}

**Here is how you find your UDID number on iPhone X, iPhone 8, iPhone 8 Plus, and everyone released before them.**

1. Launch iTunes
2. Make sure you check the box to turn off auto-syncing if you’re doing this for a co-worker.
3. Under Devices, click on your device
4. Next click on the ‘Serial Number’
5. This will change the Serial Number into the UDID

To Install haxm manually refer below link:

<https://github.com/intel/haxm/releases>