Contents

[Introduction To Appium 3](#_Toc184762714)

[Comparison between Selenium and Appium 3](#_Toc184762715)

[Features of Appium 4](#_Toc184762716)

[Advantages of Appium 4](#_Toc184762717)

[Disadvantages of Appium 5](#_Toc184762718)

[Appium Architecture 5](#_Toc184762719)

[How Does Appium Work? 6](#_Toc184762720)

[Working of Appium on Android 7](#_Toc184762721)

[Uses of Appium Framework 8](#_Toc184762722)

[Competitors of Appium 10](#_Toc184762723)

[Appium vs Robotium 11](#_Toc184762724)

[Appium vs Selendroid 11](#_Toc184762725)

[comparison between JDK and SDK 12](#_Toc184762726)

[Node.js 13](#_Toc184762727)

[Key Features of Node.js: 13](#_Toc184762728)

[Common Use Cases for Node.js: 13](#_Toc184762729)

[Types of applications 14](#_Toc184762730)

[Native applications 14](#_Toc184762731)

[Hybrid Applications 15](#_Toc184762732)

[Mobile Web Applications 15](#_Toc184762733)

[Comaparison between Mobile Application and Web Applicatoin 16](#_Toc184762734)

[Basic commands 16](#_Toc184762735)

[Adb ( Android debug bridge ) 17](#_Toc184762736)

[Mobile Testing 17](#_Toc184762737)

[Installation Testing 18](#_Toc184762738)

[Interruption Testing 18](#_Toc184762739)

[Network Dependent interruption testing 18](#_Toc184762740)

[1. Wake the Device 18](#_Toc184762741)

[2. Unlock the Device 18](#_Toc184762742)

[Functionality Dependent interruption testing 19](#_Toc184762743)

[Syntax 19](#_Toc184762744)

[Key Parameters 19](#_Toc184762745)

[Intent Format 20](#_Toc184762746)

[Device Dependent Interruption Testing 20](#_Toc184762747)

[Syntax 21](#_Toc184762748)

[Components of the Syntax 21](#_Toc184762749)

[Options 21](#_Toc184762750)

[Commonly Queried Services 21](#_Toc184762751)

[Orientation Testing 22](#_Toc184762752)

[Syntax of adb shell content 22](#_Toc184762753)

[Parameters 22](#_Toc184762754)

[Operations 22](#_Toc184762755)

[Battery Testing 22](#_Toc184762756)

[Network Testing 22](#_Toc184762757)

[Charles Proxy 22](#_Toc184762758)

[Key Features of Charles Proxy: 23](#_Toc184762759)

[GPSMobile Testing 23](#_Toc184762760)

[Gesture Testing 23](#_Toc184762761)

[Basic Syntax: 24](#_Toc184762762)

[Common Key Codes: 24](#_Toc184762763)

[StandBy Testing 24](#_Toc184762764)

[1. Simulate Screen Sleep (Turn Off Screen) 24](#_Toc184762765)

[2. Simulate Screen Wake (Turn On Screen) 24](#_Toc184762766)

[3. Force Device into Standby Mode (Sleep Mode) 25](#_Toc184762767)

[4. Simulate App Running in Background 25](#_Toc184762768)

[5. Check Battery Usage or Background Activity 25](#_Toc184762769)

[6. Monitor Device’s Network and Power State 25](#_Toc184762770)

[7. Simulate No Activity (Idle) 25](#_Toc184762771)

[8. Perform Wake-Lock Testing 26](#_Toc184762772)

[9. Monitor Wake and Sleep Transitions 26](#_Toc184762773)

[10. Simulate Doze Mode (Power Saving Mode) 26](#_Toc184762774)

[Field Testing 26](#_Toc184762775)

[UI Testing 27](#_Toc184762776)

[Uninstallation Testing 27](#_Toc184762777)

[Capabilities 27](#_Toc184762778)

[Key Differences: 28](#_Toc184762779)

[Install Appium From the Command Line 29](#_Toc184762780)

[Why to Use Appium Framework? 29](#_Toc184762781)

[ **Difference between Appium and Selendroid** 29](#_Toc184762782)

[Limitation of Appium 30](#_Toc184762783)

[Appium UiAutomator2 Driver 31](#_Toc184762784)

[Gestures 31](#_Toc184762785)

[mobile: longClickGesture 31](#_Toc184762786)

[mobile: doubleClickGesture 32](#_Toc184762787)

[mobile: clickGesture 32](#_Toc184762788)

[mobile: dragGesture 32](#_Toc184762789)

[mobile: flingGesture 33](#_Toc184762790)

[mobile: pinchOpenGesture 34](#_Toc184762791)

[mobile: pinchCloseGesture 34](#_Toc184762792)

[mobile: swipeGesture 35](#_Toc184762793)

[mobile: scrollGesture 35](#_Toc184762794)

Introduction To Appium**:**

1. **Appium** is an open-source automation mobile testing tool, which is used to test the application. It is developed and supported by **Sauce Labs** to automate native and hybrid mobile apps. It is a cross-platform mobile automation tool, which means that it allows the same test to be run on multiple platforms. Multiple devices can be easily tested by Appium in parallel.
2. Appium is used for automated testing of **native**, **hybrid**, and **web** applications. It supports automation test on the simulators (iOS) and emulators (Android) as well as physical devices (Android and iOS both).

Appium has **NO dependency** on mobile device OS because it has a framework that converts the Selenium Webdriver commands to UIAutomator and UIAutomation commands for Android and iOS respectively, that depends on the device type rather than the OS type.

|  |  |  |
| --- | --- | --- |
|  | **Appium 1** | **Appium 2** |
| 1 | Appium 1 had default android and ios support | Doesn’t have default support needs driver |
| 2 | The url for appium1 was  <http://localhost:4723/wd/hub> | The url for appium2 is <http://localhost:4723> |
| 3 | It supported both JSON Wire and W3C protocol | It supports W#C protocol |
| 4 | We were using touch action class to perform actions | Now we are using gestures to perform actions |
| 5 | In appium1 you could give any capabilities without vendor prefix | Here you can only mention automationName and browserName without vendor prefix |
| 6 | In appium1 we had Appium desktop | Now it’s called Appium inspector |
|  |  |  |

# Comparison between Selenium and Appium

Here’s a comparison between **Selenium** and **Appium**:

|  |  |  |
| --- | --- | --- |
| **Feature** | **Selenium** | **Appium** |
| **Purpose** | Web application automation | Mobile and web application automation |
| **Platform** | Desktop browsers (Chrome, Firefox, Safari, etc.) | Mobile platforms (Android, iOS) and web browsers |
| **Supported Browsers** | Chrome, Firefox, Safari, Edge, Internet Explorer | Chrome, Safari, Edge, Android WebView, iOS Safari |
| **Supported Mobile Platforms** | N/A | Android, iOS (native, hybrid, mobile web) |
| **Automation Type** | Primarily for web applications | Web, native, and hybrid mobile applications |
| **Programming Languages** | Java, Python, C#, Ruby, JavaScript, Kotlin | Java, JavaScript, Python, C#, Ruby, PHP, Node.js |
| **Set Up Complexity** | Requires browser drivers (e.g., ChromeDriver) | Requires setting up Appium server and mobile environment |
| **Mobile Testing** | Only web applications in mobile browsers | Native, hybrid, and mobile web applications |
| **Multi-Platform Support** | Cross-browser automation on desktop platforms | Cross-platform mobile testing (Android/iOS) |
| **Element Locators** | Uses XPath, CSS selectors, and other strategies | Uses UIAutomator (Android), XCUITest (iOS), XPath, and CSS |
| **Advanced Features** | Basic browser control (click, type, wait, etc.) | Device rotation, geolocation, multi-touch gestures, and camera access |
| **Parallel Test Execution** | Possible with tools like Selenium Grid | Supports multi-device testing through Appium Grid |
| **CI/CD Integration** | Strong integration with Jenkins, GitLab, etc. | Supports integration with CI/CD tools (e.g., Jenkins, GitLab) |
| **Community Support** | Large, mature community | Growing community, especially in mobile testing |
| **Use Cases** | Web automation, regression testing, performance testing | Mobile app testing (native, hybrid, mobile web) |
| **Cross-Browser Support** | Yes, supports all major browsers | Supports web browsers on mobile (Android and iOS) |
| **Test Execution Speed** | Faster on desktop browsers | Slower compared to Selenium (due to mobile emulation) |
| **Parallel Execution** | Supported via TestNG, JUnit, or Selenium Grid | Supported with Appium Grid, though more complex setup |
| **Device Interaction** | Interacts with web elements using DOM | Interacts with UI elements using the mobile device’s native API |

Here’s a comparison between **Appium** and **Calabash**, two popular tools for mobile automation testing:

| **Aspect** | **Appium** | **Calabash** |
| --- | --- | --- |
| **Definition** | Open-source automation tool for testing native, hybrid, and mobile web apps. | Open-source framework for testing native and hybrid apps with Cucumber. |
| **Programming Language** | Supports multiple languages like Java, Python, Ruby, C#, etc. | Primarily Ruby (using Cucumber syntax). |
| **Platform Support** | Cross-platform: Supports both Android and iOS. | Cross-platform: Supports both Android and iOS. |
| **Test Scripting** | Script-based approach with flexibility in language. | Behavior-driven approach (BDD) with plain English scenarios. |
| **Framework Compatibility** | Compatible with multiple testing frameworks (e.g., TestNG, JUnit). | Integrated with Cucumber for BDD-style tests. |
| **Ease of Use** | Steeper learning curve for beginners but highly flexible. | Easier for beginners due to BDD syntax, but less flexible. |
| **Architecture** | Client-server architecture using WebDriver. | Uses libraries specific to Android and iOS. |
| **Reusability** | High; reusable scripts across platforms. | Limited reusability due to Ruby-centric implementation. |
| **Setup** | Can be complex to set up and configure. | Easier setup but limited to specific scenarios. |
| **Community Support** | Strong community with wide adoption. | Smaller community with declining usage. |
| **App Type Support** | Supports native, hybrid, and mobile web apps. | Supports native and hybrid apps (no mobile web apps). |
| **Maintenance** | Actively maintained with frequent updates. | Maintenance and updates are inconsistent. |
| **Advantages** | - Wide language support.  - Strong ecosystem and community.  - No need to recompile apps. | - Easy for testers familiar with BDD. - Simplifies collaboration between technical and non-technical stakeholders. |
| **Disadvantages** | - Requires technical expertise. - Initial setup is complex. | - Limited language support. - Deprecated for iOS apps as it relies on older technologies. |

### Key Decision Points:

# Features of Appium

* Appium has multi-platform support i.e., it can run the same test cases on multiple platforms.
* Appium allows the parallel execution of test scripts.
* In Appium, a small change does not require re-installation of the application.
* Appium supports various languages like C#, Python, Java, Ruby, PHP, JavaScript with node.js, and many others that have Selenium client library.
* It is open source
* Supports multiple OS ( win ,mac ..)
* Appium is wrapper of selenium
* Underlying webdriver structure of appium is selenium
* Supports w3c protocol as like selenium
* Using appium we can automate all kind of applications

# Advantages of Appium

* Appium is an open-source tool, which means it is freely available. It is easy to install.
* It allows the automated testing of hybrid, native, and web applications.
* Integration with CI/CD(Jenkins)
* Parallel Test execution across multiple devices.
* Appium is a cross-platform, freely available mobile testing tool, which allows us the cross-platform mobile testing. This means you can test on multiple platforms (single API for both Android and IOS platforms).

# Disadvantages of Appium

* Lack of detailed reports.
* Performance issues
* Limited support for some iOS versions
* Difficult with Gestures.

# Appium Architecture

* 1. Appium server is a Node.js
  2. Appium client initiates session using capabilities (JSON Object)
  3. It drives android session using UI automator2 driver and iOS session using xcuitestdrvier.
  4. The complete appium runs on W3C protocol.

How Appium work?

* When we install the Appium, a server is also installed with it on our machine that exposes the REST API.
* It receives command and connection requests from the client and executes that command on devices like iOS or Android.
* It replies with the HTTP responses.
* To execute requests, it uses a mobile test automation framework to run the user interface of the app. For Example -
  + **Apple** instruments used for iOS
  + **Selendroid** used for Android API 15 or less
  + **UIAutomator** used for Android API 16 or higher

**Appium in Android**

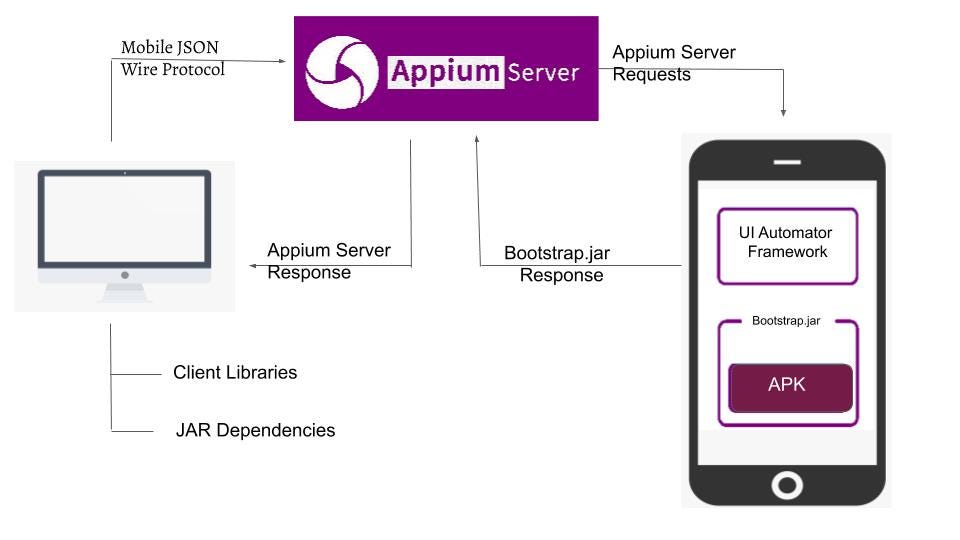
On Android, Appium proxies the command to a **UIAutomator** script running on the device. UIAutomator is a native UI automation framework of Android that allows you to run **Junit** test cases directly into the device using command line. Although it uses Java programming language, but Appium allows to run it from any WebDriver supported language.

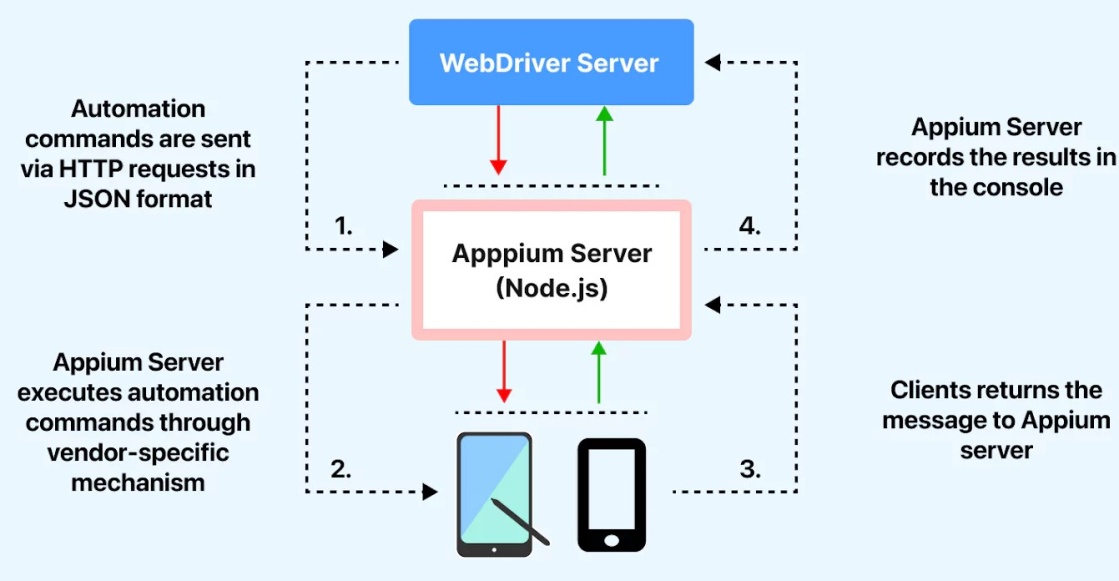
Android uses **bootstrap.jar**, which works as a TCP server. It is used to send the test commands to perform the actions on Android device using UIAutomator.

In the below figure, see the Appium architecture in respect to Android automation -

## How Does Appium Work?

The client interacts with the Appium server through REST APIs, which are controlled by the Mobile JSON Wire Protocol. The REST APIs accept client connections, listen for commands, execute commands, and send back the command execution status.



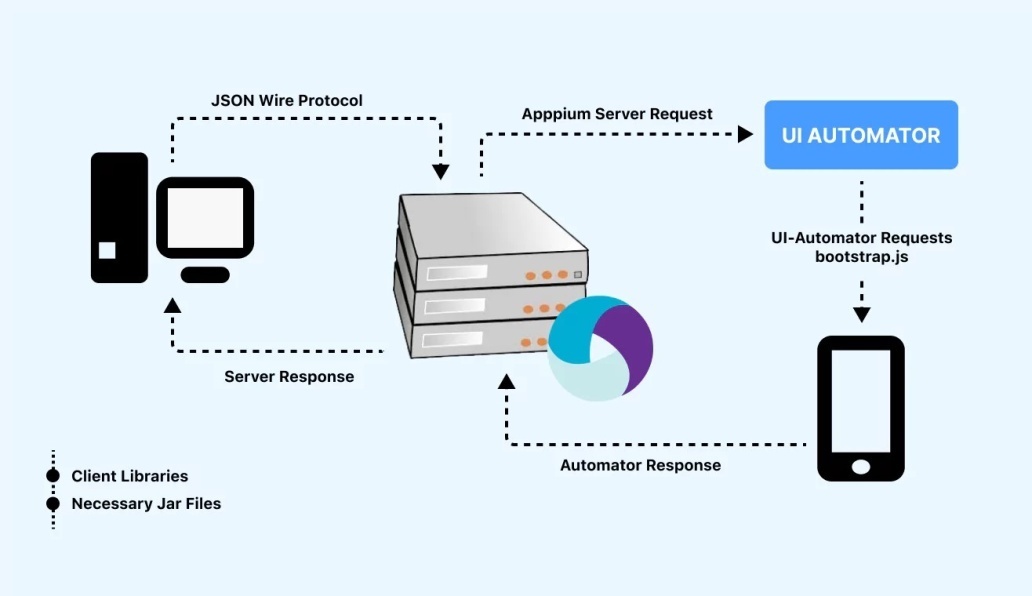


1. The first step to perform Appium testing is to create a session.
2. During a session, the client sends a request to the server containing session-related information in key-value pairs, also known as Desired Capabilities. The Desired Capabilities enable Appium to differentiate between platforms (iOS or Android) and fire up a session on the mobile device, Android Emulator, or iOS Simulator.
3. Using the JSON object (Desired Capabilities), the POST request is sent to the server, and Appium responds to it in the form of a session ID.
4. Once the session is up, the client and Appium server can interact with each other.

## Working of Appium on Android

Appium Client Libraries convert the test commands to the REST API requests with the help of the Mobile JSON Wire Protocol. The Appium server forwards these requests to the Android device or emulator. The Android device contains the bootstrap.jar files. The bootstrap.jar file interprets these test commands. The bootstrap.jar file uses Selendroid or UI Automator to execute the requests on the Android device.

The results of the tests are then sent to the Appium server, which sends an HTTP response (containing status codes) to the Appium client.



***What is the JSON Wire Protocol?***

Communication between client and server via REST API takes place in the form of exchange of JSON(JavaScript Object Notations).

*JSON is a lightweight, language independent data interchange format.*

Example of basic JSON:

{  
"Student":{  
"FirstName":"Appium",  
"LastName":"Selenium",  
"IdNumber":"12345",  
"City" : "New Delhi",  
"EmailID" : "email@gmail.com" }  
}

***JSON Wire Protocol*** is a[*predefined set of specifications*](https://w3c.github.io/webdriver/) that maps actions such as click, type, scroll etc with the HTTP Request/Response. In simple terms, they are a set of rules that define what data should be sent in, in what order, and in what format between client and server.

Appium uses ***Mobile JSON Wire Protocol***, which extends JSON Wire Protocol. It enables Appium Server to manage communication with Mobile Devices.

**Flow of communication between Client and Server**

A client wants to perform an action on the device. So it converts the action, as object, into JSON object and sends it to the server. The server parses the JSON object and converts it to object. Now server process this object and converts the response object into JSON Object and sends it back to the client. The client then converts the JSON Object to the object.

***What is Bootstrap.jar?***

Appium server interacts with Android devices through bootstrap.jar. When server starts an Android driver session, it pushes ***bootstrap.jar*** file to the device. Device executes this file using the device’s built-in uiautomator command. When bootstrap.jar is executed by the device, it starts a server that listens on the **port 4724**. This server listens for the requests that are coming from the Appium Server.

On receiving the command, it converts them into UIAutomator commands, understandable by Android API 17 or higher. This UIAutomator then performs the desired action on the device.

***What is WebDriverAgent.app?***

WebDriverAgent is a ***WebDriver server*** implementation for the iOS that can be used to remote control iOS devices. It enables to perform actions such as launch and kill applications, tap, scroll views etc. It works by linking XCUITest.framework and calling Apple’s API to execute commands directly on a device/simulator. WebDriverAgent is developed and used at **Facebook** for end-to-end testing and has been successfully adopted and integrated with Appium for iOS backed XCUITest Framework.

When Appium first interacts with an iOS device/simulator, it checks for WebDriverAgent.app. If the app is not present on the device then it installs the WebDriverAgent.app as the primary step

# Uses of Appium Framework

The Appium framework Use client and server components. The client initiates the test commands and send them to the server. The server component (Appium server) receives commands from the client, translates them into actions, and executes them on the device or emulator.Communication Protocol between the Client and the Server

* Appium relies on the [JSON](https://www.geeksforgeeks.org/json-data-types/) wire protocol or the newer WebDriver protocol for communication between the client and server.
* The client sends [HTTP requests](https://www.geeksforgeeks.org/different-kinds-of-http-requests/) to the Appium server, specifying desired capabilities and commands.
* The server interprets these requests, interacts with the mobile device or emulator, and sends back responses to the client.
* This protocol ensures seamless communication between the testing script and the automation engine, facilitating efficient test execution.

**Advantages of Appium**

* Supports automated testing across multiple platforms
* Freely available and flexible.
* Provides wide language support that includes Java, [Python](https://www.geeksforgeeks.org/python-programming-language/), C# and more that enable tests in the desired language.
* Can automate testing for native and hybrid model applications
* Does not require any modification of the app code.

**Disadvantages of Appium**

* Relies on third party tools like UI Automator, XCUITest.
* Limited support for non-mobile platforms such as web applications/
* Slow execution speed compared to the native testing frameworks.
* Setting up Appium can be complex and time-consuming.
* Test scripts may become brittle and require frequent maintenance.

# Competitors of Appium

There are several tools available for automated testing mobile applications, such as Robotium, Appium, Experitest, Selendroid, Kobiton, and Testdroid, etc. They all are tough contestants for Appium. But Selendroid and Robotium are one of the top competitors of Appium. Let us know some differences and see how they differ from each other.

## Appium vs Robotium

* **Appium** is a cross-platform tool that supports both iOS and Android. Whereas **Robotium** only supports Android.
* **Appium** supports various languages while **Robotium** only supports Java programming language.
* **Appium** does not require application source code/library, whereas **Robotium** tool requires application source code or library.
* **Appium** can be used to test native, web, and hybrid mobile applications, whereas **Robotium** can only test native and hybrid applications.
* **Appium** supports many frameworks like Selenium. But **Robotium** is not compatible with Selenium at all.
* **In Appium**, you don't have to reinstall the application for a small change. But **Robotium** code leads to complete rebuild for a small change.

## Appium vs Selendroid

* **Appium** is an open-source automation tool that supports both iOS and Android, while **Selendroid** is a test automation framework that only supports Android.
* In **Appium**, a small change does not require reinstallation of the application. But **Selendroid** requires reinstallation of the application.
* **Appium** has a strong and active community, whereas **Selendroid** does not have a strong community like Appium.
* **Appium** supports many frameworks and languages. On the other hand, **Selendroid** is compatible with Jenkin and Selenium.
* **Appium** does not require application source code/library, while **Selendroid** requires application source code or library.
* **Appium** supports all Android APIs with a limitation. Appium uses UIAutomator for tests running on API>=17, while for older APIs, it runs tests using Selendroid.

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Appium** | **Selendroid** |
| **Platform Support** | Appium offers cross-platform support, making it suitable for Android and iOS projects | Selendroid primarily focuses on Android applications, making it ideal for Android-only projects. |
| **Native and Hybrid App Testing** | Appium is capable of testing mobile, hybrid, and native web applications, supporting a wide range of app types. | Selendroid also supports mobile, hybrid, and native web applications, offering similar capabilities. |
| **Automation Language** | Appium supports multiple programming languages (Java, Python, Ruby, C#, etc.), providing flexibility for testers with different language preferences. | Selendroid primarily relies on Java for test automation, limiting options for testers who prefer other languages |
| **Test Execution** | Appium allows tests to run on simulators, emulators, and actual hardware, providing flexibility in choosing the testing environment. | Selendroid supports test execution on real Android devices but lacks built-in support for iOS simulators. |
| **Continuous Development** | Appium undergoes regular updates, ensuring compatibility with the latest mobile platforms and technologies. | Selendroid's development has slowed down, potentially limiting its ability to support the newest Android features and devices. |
| **Ecosystem and Community** | Appium boasts a large and active user community, offering a wealth of resources, tutorials, and support. It also features a wide ecosystem of plugins and integrations. | Selendroid has a user base and community but is not as widely adopted as Appium, potentially resulting in fewer available resources and community support. |
| **Testing Strategy** | Appium is a versatile choice suitable for various mobile testing scenarios, accommodating cross-platform testing, multiple programming languages, and a broad range of app types. | Selendroid is specialized for automating native Android apps, excelling in Android-specific testing but not suitable for cross-platform or iOS testing. |

# comparison between JDK and SDK

|  |  |  |
| --- | --- | --- |
| **Feature** | **JDK (Java Development Kit)** | **SDK (Software Development Kit)** |
| **Definition** | A toolkit specifically for developing Java applications. | A general toolkit for developing applications for a specific platform or technology. |
| **Scope** | Limited to Java development. | Broader, applicable to various programming languages and platforms. |
| **Components** | Includes JRE, javac (compiler), java (launcher), libraries, etc. | May include APIs, libraries, tools, documentation, and examples for a specific platform. |
| **Platform Dependency** | Platform-independent; runs on any system with Java installed. | Platform-specific; varies based on the platform (e.g., Android SDK, Windows SDK). |
| **Purpose** | To write, compile, and run Java applications. | To develop applications tailored for a specific platform or ecosystem. |
| **Examples** | OpenJDK, Oracle JDK, Amazon Corretto. | Android SDK, iOS SDK, Windows SDK. |
| **Use Case** | Java application development. | Application development for specific technologies or platforms. |

**Android Sdk:**

* 1. To develop UI we need android features android.button.view
  2. Deploy to customer we need **android SDK**
  3. **Components of SDK**
     1. **Build Tool** => Build Binaries => .**apk ( android package Kit)**
        + Converting source code into executable format is called build binaries
     2. **Android Emulator:** It is virtual machine like real machine
     3. **Platform tools**
        + It shows any error in the project to debug the app
        + **Adb => it is commandline tool which is used to debug the app**
     4. **Platform**-> versions of android.

### Android Versions

|  |  |  |
| --- | --- | --- |
| **Version** | **Code Name** | **Release Date** |
| 1.0 | No Code Name | September 2008 |
| 1.1 | Petit Four (unofficial) | February 2009 |
| 1.5 | Cupcake | April 2009 |
| 1.6 | Donut | September 2009 |
| 2.0-2.1 | Eclair | October 2009 |
| 2.2 | Froyo (Frozen Yogurt) | May 2010 |
| 2.3 | Gingerbread | December 2010 |
| 3.0-3.2 | Honeycomb (tablet-focused) | February 2011 |
| 4.0 | Ice Cream Sandwich | October 2011 |
| 4.1-4.3 | Jelly Bean | July 2012 |
| 4.4 | KitKat | October 2013 |
| 5.0-5.1 | Lollipop | November 2014 |
| 6.0 | Marshmallow | October 2015 |
| 7.0-7.1 | Nougat | August 2016 |
| 8.0-8.1 | Oreo | August 2017 |
| 9 | Pie | August 2018 |
| 10 | Android 10 | September 2019 |
| 11 | Android 11 | September 2020 |
| 12 | Android 12 | October 2021 |
| 13 | Android 13 | August 2022 |
| 14 | Android 14 | October 2023 |

**Appium Inspector** is a tool used for inspecting and interacting with the user interface (UI) of mobile applications in an automated testing environment. It is an essential part of the **Appium** ecosystem, used for interacting with mobile apps through Appium’s capabilities, such as for inspecting the app’s elements, generating selectors, and debugging automation scripts.

Here’s a detailed overview:

### ****Key Features of Appium Inspector****:

1. **UI Element Inspection**:
   * Allows you to inspect the UI of an app, making it easier to understand and interact with the app’s elements.
   * Displays a **hierarchical view** of the elements in the application (buttons, text fields, images, etc.), and helps identify the attributes (e.g., ID, class, XPath, etc.).
2. **Element Locators**:
   * Automatically generates **locators** (XPath, ID, Name) based on the app’s UI.
   * Helps with **Element Identification** to create automation scripts effectively.
   * **Save Locators**: It saves the generated locators for later use in automation scripts.
3. **Appium Server Connection**:
   * Works by connecting to an **Appium server** that communicates with the mobile device or emulator.
   * Can connect to **iOS** and **Android** applications for inspection and automation.
   * Allows you to interact with both **native** and **hybrid** apps.
4. **Recording Interactions**:
   * **Record actions**: The tool can record interactions with the app (e.g., clicks, scrolls) and replay them.
   * Helps in automating tasks like form filling, clicking buttons, and navigating screens.
5. **Supports Multiple Devices**:
   * Can be used to inspect and interact with multiple devices, emulators, and simulators simultaneously.
   * Supports both **Android** and **iOS** apps.
6. **Session Control**:
   * You can start, stop, and reset Appium sessions from the Inspector UI.
   * Helps in debugging your automation tests by viewing real-time interactions between your test and the app.
7. **Appium’s WebDriver Commands**:
   * Executes **WebDriver commands** and shows results in real-time.
   * Provides **debugging** for mobile apps by sending actions directly to the app (e.g., clicking buttons, entering text).
8. **Real-Time App Interaction**:
   * Allows you to **simulate** taps, swipes, and other gestures on the mobile app interface, interacting with the app's UI in real-time.
   * Facilitates automation testing by verifying app behaviors after performing certain actions.

### ****How Appium Inspector Works****:

* **Step 1: Appium Setup**: You need an Appium server running. You can either use the Appium desktop application or run Appium from the command line.
* **Step 2: Device/Simulator/Emulator Setup**: The mobile device or simulator/emulator should be connected and running the app you want to inspect.
* **Step 3: Launch Appium Inspector**: Open Appium Inspector and connect it to your Appium server.
* **Step 4: Connect to the App**: Provide the necessary desired capabilities (like platform name, device name, app path, etc.) to connect to the app.
* **Step 5: Inspect the App**: Once connected, the Appium Inspector displays the app’s UI, and you can start inspecting and interacting with the elements.

# Node.js

**Node.js** is a powerful, open-source runtime environment that allows you to run JavaScript on the server-side (outside of a browser). It is built on Chrome's V8 JavaScript engine and is designed to be efficient, lightweight, and scalable, making it a popular choice for building server-side applications, particularly for real-time, I/O-heavy tasks.

Here are the key aspects of Node.js and its use cases:

### ****Key Features of Node.js:****

1. **Asynchronous and Event-Driven**:
   * Node.js uses non-blocking, event-driven architecture, making it highly efficient for I/O-bound operations like reading files or handling multiple network requests simultaneously.
2. **Single-Threaded**:
   * Node.js operates on a single thread using asynchronous calls, making it capable of handling many connections concurrently. It avoids the overhead of thread creation and management typically found in multi-threaded servers.
3. **Cross-Platform**:
   * Node.js works on major operating systems like Windows, Linux, and macOS, making it versatile for different environments.
4. **Fast Execution (V8 Engine)**:
   * Node.js uses Google's V8 JavaScript engine, which is highly optimized for performance, enabling fast execution of JavaScript code.
5. **NPM (Node Package Manager)**:
   * Node.js has an extensive ecosystem with **npm**, the largest package manager for JavaScript. You can install packages (libraries, tools) easily and integrate them into your projects.
6. **Built-In Libraries**:
   * Node.js comes with built-in libraries like **http**, **fs** (for filesystem operations), **path**, and **os** that help developers build applications quickly without needing third-party dependencies.

### ****Common Use Cases for Node.js****:

1. **Web Servers and APIs**:
   * Node.js is commonly used to build fast and scalable web servers and RESTful APIs. For instance, you can use the **Express.js** framework to create server-side applications.
2. **Real-Time Applications**:
   * Due to its asynchronous nature, Node.js is ideal for building real-time applications like chat apps, online gaming, or collaborative tools (e.g., Google Docs).
3. **Microservices**:
   * Node.js is often used to build lightweight microservices due to its small footprint and fast execution. It’s particularly popular for handling high volumes of requests in microservices architectures.
4. **Automation and Scripting**:
   * Node.js is commonly used for writing automation scripts, such as for testing frameworks (e.g., Appium for mobile testing), task automation, or cron jobs.
5. **File I/O**:
   * Node.js is well-suited for applications that need to handle large amounts of data, such as streaming applications or those that require reading and writing large files asynchronously.

Appium is typically built on **Node.js**, and to run Appium, you need Node.js installed. When working with Appium, you’ll often interact with Node.js in the context of:

1. **Setting up and running the Appium server** using Node.js.
2. **Creating custom scripts** or **client libraries** in JavaScript to interact with Appium, as Appium provides a JavaScript client binding

Appium’s Node.js client allows you to execute a wide range of commands for interacting with mobile devices, including:

* Sending ADB commands
* Clicking buttons
* Typing text
* Swiping, scrolling, etc.

To install Appium => npm install -g Appium@2.5.4

>appium -v

>appium driver install uiautomator2

>

# Types of applications

## Native applications

* 1. Native application are the applications developed in native language for specific platforms.
  2. Android app is developed using java / kottlin
  3. iOS App is developed using Swift/ obj.c
  4. The native application are faster compared to hybrid application (performance)
  5. Native application will have lot of dependencies command over native features
  6. Android SDK is used to develop android native application ( android studio IDE) and Xcode for iOS applications
  7. UI of native application is not attractive and it is user friendly application

**Advantages**

* + 1. Native applications are good in speed and performance since it is developed in native language.
    2. They will have good command over mobile hardware and software components
    3. They are secured when compared to hybrid or web application
    4. Simple UI hence enhancing user experience
    5. Can work with low internet or offline as well.

**Disadvantages**

1. For native applications the source code is maintained separetly because the language used is different, hence the cost of maintainence has more.
2. We also require separate teams for developing android and iOS applications .
3. Modification or updation of application and releasing is difficult since it has to done twice.

## Hybrid Applications

1. Hybrid applications are web application which is wrapped inside native container.
2. These are the combinations of web and native application we can say it as web application in a native container
3. These are not platform specific , same source code will work for both android and iOS because developer use common language like HTML , CSS , react native , flutter to develop these applications.
4. Few hybrid applications will have in built browsers

**Advantage**

* + 1. We can have same source since we use common code for both platforms by using react-native languages which will give 70-80 % of similarity and only 20-30% of the code changes due to dependency of native content, now days flutter is introduced and almost 95% of code is common for all the platforms a which includes android, iOS , mobile web and web application.
    2. Hence the cost and code maintenance is less
    3. One team can manage developing and testing of application as the code is almost same.
    4. Modification and deployment is easier.

**Disadvantages**

1. Performance is slower.
2. Limited access to the native features.
3. Dependent on internet speed and third party API’s
4. Can lead to compatibility issue.

## Mobile Web Applications

1. An application which we can access through our mobile browser is known as mobile web applications
2. Here are have two types
   1. **Simple Web applications**
      1. These are built for desktop browser but can be accessed through mobile browsers as well. Example : **EPFO** , college university, DL or any govt . websites
   2. **Progressive Web applications**
      1. These are built separately for desktop browser and mobile browsers .
      2. These applications UI will differ when accessed in mobile from web. Example : facebook.com

## Comaparison between Mobile Application and Web Applicatoin

|  |  |  |
| --- | --- | --- |
| **SLNO** | **MOBILE APPLICATION** | **WEB APPLICATION** |
| 1 | Runs on mobile devices | Runs on Web browsers |
| 2 | Needs to downloaded and installed from an app store | Does not require installation accessed via URL |
| 3 | Accessed only through the devices app icon | Accessible through any device with a browser and internet connection |
| 4 | Generally faster and more responsive as they run natively on the devices | May be slower due to reliance on the internet and browser performance |
| 5 | Requires users to download update via the app store | Automatically updated when changes are made on the server side. |
| 6 | Distributed through app stores with approval process (playstore) | Directly accessibility by anyone with a URL without app store restrictions. |

# Basic commands

* 1. To install appium
     + >npm install –g [appium@2.5.4](mailto:appium@2.5.4)
  2. To find appium version
     + >appium –v
  3. To install the driver
     + >appiium driver install uiautomator2
  4. To check list of driver installed
     + >appium driver list

Listing available drivers

* + - - **uiautomator2@3.9.0 [installed (npm)]**
    - - xcuitest [not installed]
    - - mac2 [not installed]
    - - espresso [not installed]
    - - safari [not installed]
    - - gecko [not installed]
    - - chromium [not installed]
  1. To start the appium server
     + >appium

# Adb ( Android debug bridge )

1. It is command line tool that is used to **communicate with a device.** The **adb** command facilitates a variety of device actions such as installing and debugging apps.
2. It is client-server program that includes three components
   1. **A client** **which sends commands** . The client runs on development machine. You can invoke a client from command-line terminal by issuing an adb command.
   2. **A daemon(adbd),** **which runs commands on device**. The daemon runs as a background process on each device.
   3. **A server**, which manages communication between client and daemon. The server runs as a background process on your development machine.

To check the **ADB version** installed on your system, you can use the following command:

### Command:

bash

Copy code

adb version

### Example Output:

text

Copy code

Android Debug Bridge version 1.0.41

Version 33.0.3-8952118

Installed as /path/to/adb

### What the Output Means:

1. **ADB Version**: The current version of the Android Debug Bridge installed on your system.
2. **Build Number**: A specific build identifier for that version.
3. **Installed Path**: The directory where ADB is installed.

**How adb works**

* 1. When you start an adb client, the client first checks whether there is an adb server process already running
  2. If there isn't, it starts the server process. When the server starts, it binds to local TCP port 5037 and listens for commands sent from adb clients.

**Note:** All **adb** clients use port 5037 to communicate with the **adb** server.

* 1. The server then sets up connections to all running devices.
  2. It locates emulators by scanning odd-numbered ports in the range 5555 to 5585, which is the range used by the first **16 emulators**. Where the server finds an adb daemon (adbd), it sets up a connection to that port.
  3. Each emulator uses a pair of sequential ports — an even-numbered port for console connections and an odd-numbered port for adb connections.
  4. For example:

Emulator 1, console: 5554  
Emulator 1, adb: 5555  
Emulator 2, console: 5556  
Emulator 2, adb: 5557  
and so on.

* 1. As shown, the emulator connected to adb on port 5555 is the same as the emulator whose console listens on port 5554.
  2. Once the server has set up connections to all devices, you can use adb commands to access those devices. Because the server manages connections to devices and handles commands from multiple adb clients, you can control any device from any client or from a script.

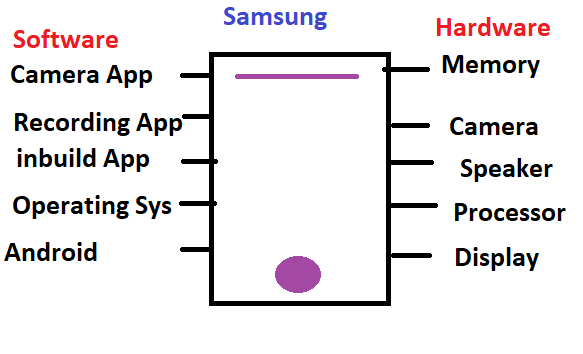
**> appium – -allow-insecure chromedriver\_autodownload**

The **--allow-insecure chromedriver\_autodownload** flag is used with **Appium Server** to allow Appium to automatically download compatible versions of ChromeDriver for automating web-based content in Chrome browsers or WebView contexts.

**>adb devices**

# Mobile Testing

* 1. Testing the mobile application or device is called mobile testing.
     + Mobile device testing- Testing hardware and software device is called mobile device testing.



* + - Mobile Application Testing- Testing the mobile application into the device.

## Installation Testing

* 1. Installing the application after several time uninstallation
  2. Copy apk file =>apdemos.debug.apk
  3. >adb devices.
  4. >adb install “give path of apk files”.

## Interruption Testing

1. Testing the interruption of an application we call it as interruption testing.
2. Check the application whether it is working fine after the disturbance

### Network Dependent interruption testing

* + 1. Try to install application in wifi and disable in between.
    2. **Commond:** 
       1. **Syntax****: adb shell svc <service><action><option>**
          1. **>adb shell svc wifi disable**
          2. **>adb shell svc wifi enable**
          3. **>adb shell svc data/Bluetooth enable/disable**
          4. **>adb shell svc power stayon true/false/usb**

### ****1. Wake the Device****

If the device screen is off, wake it up using:

**adb shell input keyevent 224**

(Keyevent 224 corresponds to the **POWER** button.)

### Power & System Controls:

* KEYCODE\_POWER (26) - Power button.
* KEYCODE\_SLEEP (223) - Sleep button.
* KEYCODE\_WAKEUP (224) - Wake-up action.
* KEYCODE\_VOLUME\_UP (24) - Volume Up button.
* KEYCODE\_VOLUME\_DOWN (25) - Volume Down button.
* KEYCODE\_VOLUME\_MUTE (164) - Mute button.

### Navigation Controls:

* KEYCODE\_HOME (3) - Home button.
* KEYCODE\_BACK (4) - Back button.
* KEYCODE\_MENU (82) - Menu button.
* KEYCODE\_APP\_SWITCH (187) - App switch (Recent Apps) button.

### Media Controls:

* KEYCODE\_MEDIA\_PLAY (126) - Play media.
* KEYCODE\_MEDIA\_PAUSE (127) - Pause media.
* KEYCODE\_MEDIA\_PLAY\_PAUSE (85) - Toggle Play/Pause.
* KEYCODE\_MEDIA\_STOP (86) - Stop media.
* KEYCODE\_MEDIA\_NEXT (87) - Next track.
* KEYCODE\_MEDIA\_PREVIOUS (88) - Previous track.
* KEYCODE\_MEDIA\_FAST\_FORWARD (90) - Fast forward.
* KEYCODE\_MEDIA\_REWIND (89) - Rewind.

### Call & Communication:

* KEYCODE\_CALL (5) - Call button.
* KEYCODE\_ENDCALL (6) - End call button.
* KEYCODE\_CAMERA (27) - Camera button.

### Text Input & Keyboard:

* KEYCODE\_A to KEYCODE\_Z (29 to 54) - Alphabet keys.
* KEYCODE\_0 to KEYCODE\_9 (7 to 16) - Number keys.
* KEYCODE\_ENTER (66) - Enter key.
* KEYCODE\_DEL (67) - Delete key.
* KEYCODE\_TAB (61) - Tab key.
* KEYCODE\_SPACE (62) - Space key.

### Directional & Navigation Keys:

* KEYCODE\_DPAD\_UP (19) - Up direction.
* KEYCODE\_DPAD\_DOWN (20) - Down direction.
* KEYCODE\_DPAD\_LEFT (21) - Left direction.
* KEYCODE\_DPAD\_RIGHT (22) - Right direction.
* KEYCODE\_DPAD\_CENTER (23) - DPAD Center click.

### Miscellaneous:

* KEYCODE\_ESCAPE (111) - Escape key.
* KEYCODE\_F1 to KEYCODE\_F12 (131 to 142) - Function keys.
* KEYCODE\_SEARCH (84) - Search button.
* KEYCODE\_CAPS\_LOCK (115) - Caps Lock key.
* KEYCODE\_NUM\_LOCK (143) - Num Lock key.

### Special Keys:

* KEYCODE\_MEDIA\_RECORD (130) - Record button.
* KEYCODE\_SETTINGS (176) - Settings key.
* KEYCODE\_ASSIST (219) - Assistant button

### ****2. Unlock the Device****

You can unlock the device using one of the following methods:

#### **2.1 Swipe to Unlock**

If the device has a swipe gesture to unlock:

**adb shell input swipe <x1> <y1> <x2> <y2> [duration]**

**adb shell input swipe 300 1000 300 500**

This simulates a swipe gesture. Adjust the coordinates (300 1000 to 300 500) based on the device's screen resolution.

#### **2.2 Enter a PIN/Password**

If the device is locked with a PIN or password, send it using:

adb shell input text "<your\_PIN\_or\_password>"

adb shell input keyevent 66

(Keyevent 66 simulates the **ENTER** key.)

Example for a PIN 1234:

adb shell input text 1234

adb shell input keyevent 66

**Tap**:

adb shell input tap <x> <y>

Example:

adb shell input tap 500 800

### Functionality Dependent interruption testing

1. The interruption which is called one application to another application.
2. An application getting interrupted by another application is called functionality dependent interruption testing
3. **Command :**
   * + 1. syntax:
       2. >adb shell am start –a android.intent.action.CALL –d <tel:+91XXXXXXX>
       3. >adb shell am start –a android.intent.action.SENDTO –d sms:XXXXXX

--es sms\_body “your\ message” –ez exit\_on\_Sent true.

### ****Syntax****

**adb shell am start [-D] [-W] [-P <PROFILER\_FILE>] [-R <COUNT>]**

**[-S] [--user <USER\_ID>] <INTENT>**

### ****Key Parameters****

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| -D | Enables debugging mode. |
| -W | Waits for the launch to complete before returning. |
| -P <PROFILER\_FILE> | Specifies a file for profiling output. |
| -R <COUNT> | Repeats the launch a specified number of times. |
| -S | Forces the activity to stop before starting. |
| --user <USER\_ID> | Specifies the user to run the command as (multi-user environment). |
| <INTENT> | Defines the intent that specifies the target activity or application. See below for details. |

### ****Intent Format****

The intent specifies the target activity and optional data or actions. The typical components are:

1. **Component (Required)**: Specifies the package and activity.

-n <PACKAGE\_NAME>/<ACTIVITY\_NAME>

1. **Action (Optional)**: Specifies the action to perform.

-a <ACTION>

1. **Data URI (Optional)**: Specifies data to pass.

-d <DATA\_URI>

1. **MIME Type (Optional)**: Specifies the type of data.

-t <MIME\_TYPE>

1. **Extras (Optional)**: Passes additional key-value pairs.

--es <KEY><STRING\_VALUE>

--ei<KEY><INT\_VALUE>

### Device Dependent Interruption Testing

1. The interruption calls by the devices functionality is device dependent interruption testing.
2. Commands:
   * + 1. >adb shell dumpsys battery set level 5
       2. >adb shell dumpsys battery set level 100
       3. >adb shell dumpsys batter set usb 0 ( device usb disconnected )
       4. >adb shell dumpsys battery set level 5
       5. >adb shell dumpsys battery set level 1
       6. >adb shell dumpsys battery set level 0
       7. >adb shell dumpsys battery set level 100 ( charging)
       8. >adb shell dumpsys battery set usb 1
       9. >adb shell dumpsys battery reset
       10. >adb shell input Keyevent KEYCODE\_HOME ( while playing youtube video-> go back.
       11. >adb shell input Keyevent KEYCODE\_BACK

### ****Syntax****

**adb shell dumpsys [OPTIONS] [SERVICE]**

### ****Components of the Syntax****

|  |  |
| --- | --- |
| **Component** | **Description** |
| **OPTIONS** | Optional arguments to refine the output, like specifying filters or formats. |
| **SERVICE** | Specifies the system service to query. If omitted, dumpsys provides information about all services. |

### ****Options****

|  |  |
| --- | --- |
| **Option** | **Description** |
| -h | Displays help for the dumpsys command. |
| -l | Lists all available services that can be queried. |
| --proto | Outputs the dump in protobuf format (if supported by the service). |
| --checkin | Outputs a more compact format suitable for programmatic parsing. |

### ****Commonly Queried Services****

|  |  |
| --- | --- |
| **Service** | **Description** |
| activity | Information about activities, tasks, and processes. |
| window | Window manager state (e.g., active windows, display info). |
| package | Details about installed packages and their permissions. |
| wifi | Wi-Fi state and configuration. |
| battery | Battery state, charging status, and health. |
| diskstats | Disk usage statistics. |
| cpuinfo | CPU usage and performance metrics. |
| input | Input devices and their states. |
| media.audio\_policy | Audio routing and output configuration. |
| telecom | Phone call and telephony states. |

**dumpsys** stands for **"Dump System"**. It is used to retrieve detailed information about various system services on an Android device.

 It outputs a variety of system data, including service status, device health, resource usage, logs, and more.

## Orientation Testing

1. Checking the orientation of mobile devices ( Portrait and Landscape)
2. **Command:** 
   * 1. >adb shell content insert –uri content://settings/system –bind name:s:accelerometer\_rotation –bind value:i:0
     2. 0 –disable, portrait ( 2)
     3. 1- enable, landscape(3)

### ****Syntax of**** adb shell content

adb shell content [<operation>] <URI> [<args>]

### ****Parameters****

* **<operation>**: The operation to perform on the content provider (e.g., query, insert, update, delete).
* **<URI>**: The content URI identifying the data to operate on.
* **<args>**: Additional arguments, depending on the operation (e.g., columns, values, where clause, etc.).

### ****Operations****

* **query**: Retrieves data from the content provider.
* **insert**: Inserts data into the content provider.
* **update**: Updates existing data in the content provider.
* **delete**: Deletes data from the content provider.

## Battery Testing

1. Testing the application how much battery it is consuming is known as battery consumption testing.
2. Behaviour of application based on different battery level.
3. Battery level is low – power saving mode
4. Ex: video calling

## Network Testing

1. Test the application in different network is called network testing.
2. Charles proxy: change speed of network, throttle of network to test the application.

## **Charles Proxy**

1. **Charles Proxy is a web debugging tool that acts as a HTTP proxy server, enabling you to inspect all the HTTP and HTTPS traffic between your computer (or mobile device) and the internet.**
2. **It is widely used for web development, mobile app testing, and debugging API requests.**
3. **Charles Proxy allows you to view the details of requests and responses, modify them in real-time, and analyze performance.**

### ****Key Features of Charles Proxy:****

1. **HTTP/HTTPS Proxy**: Charles intercepts and logs all HTTP and HTTPS traffic between your computer/device and the internet.
2. **SSL Proxying**: It can decrypt SSL-encrypted traffic (HTTPS) by acting as a "man-in-the-middle" proxy.
3. **Request/Response Inspection**: You can inspect request headers, body, responses, cookies, and more.
4. **Throttling**: Simulate slow network connections to test how your app or website performs under different conditions.
5. **Breakpoints**: Set breakpoints to pause traffic and modify requests or responses before they are sent or received.
6. **WebSocket Support**: Charles also supports WebSocket traffic, so you can inspect WebSocket messages.
7. **Session Recording**: It allows you to record sessions and save them for later analysis.
8. **AJAX Debugging**: Helps debug and monitor AJAX calls made by web applications.
9. **Bandwidth Simulation**: Test how your application performs on slow, fast, or even unreliable networks.
10. **Advanced Filters**: Allows for filtering and searching for specific requests/responses.

## GPSMobile Testing

1. Testing the application based on geographical location are called as GPS Testing.
2. Install fake GPS app in mobile.
3. Change the location some other place and click on change
4. Developer option from setting and choose fake location (Scorlldown to bottom)
5. Check in google map whether location is changed or not
6. Try to orderzomato and will get to know location changed.

## Gesture Testing

1. Scroll up & down , click action, drag&drop, 2 finger swipe , 3 finger swipe are called Gesture action in mobile.
2. Testing these action in mobile device is called gesture testing
3. **Commands** 
   * 1. >adb shell input tab 500 1000
     2. >adb shell input swipe 400 500 400 500
     3. Long press
        1. >adb shell input swipe 300 500 300 500 2000
     4. Back button
        1. >adb shell input Keyevent 4
     5. Home Button
        1. >adb shell input Keyevent 3
     6. Menu Button
        1. >adb shell input keyevent 82
     7. Input text
        1. >adb shell input text “ yourtext”

### ****Basic Syntax****:

|  |  |
| --- | --- |
| **Key Code** | **Description** |
| 4 | BACK button |
| 3 | HOME button |
| 24 | VOLUME\_UP button |
| 25 | VOLUME\_DOWN button |
| 26 | POWER button (screen on/off) |
| 66 | ENTER button |
| 82 | MENU button |
| 85 | PLAY\_PAUSE button |

**adb shell input <command><parameters>**

### ****Common Key Codes****:

Here are some common key codes used with keyevent:

## StandBy Testing

1. Open the phone and we are not doing any action is called stand by

To perform **standby testing** (testing scenarios where the device is idle or in a low-power state) using ADB commands, you can simulate various actions that test the device’s behavior in standby or background conditions. These actions typically involve simulating screen sleep, wake, or testing how apps and services behave when the screen is off or the device is not actively being used.

Here are some **ADB commands** that can help with testing standby scenarios on Android devices:

### ****1. Simulate Screen Sleep (Turn Off Screen)****

To simulate the device going into standby or the screen turning off, use the following command:

**adb shell input keyevent 26**

* This command simulates pressing the **Power** button to turn the screen off.

Alternatively, you can use:

**adb shell svc power stayonfalse**

* This disables the "stay awake" mode, which ensures the screen will turn off.

### ****2. Simulate Screen Wake (Turn On Screen)****

To simulate the device waking up or turning on the screen, use this command:

adb shell input keyevent 26

* This simulates pressing the **Power** button again to turn the screen back on.

Alternatively, use:

adb shell svc power stayontrue

* This keeps the screen on (useful if you want to simulate standby without turning off the screen).

### ****3. Force Device into Standby Mode (Sleep Mode)****

To simulate the device entering a standby or sleep mode (where the system remains active but the screen is off), use:

adb shell dumpsys power | grep mWakefulness

* This command checks the current wakefulness state. If it shows **Asleep**, the device is in standby or sleep mode. You can manually turn the screen off as shown earlier or use this command to ensure the device enters standby.

### ****4. Simulate App Running in Background****

To simulate apps running in the background, you can launch an app and then minimize it (send it to the background) by pressing the **Home** button:

adb shell input keyevent 3

* This simulates pressing the **Home** button, which typically sends the app to the background.

### ****5. Check Battery Usage or Background Activity****

To monitor battery usage and the impact of background activities, you can use:

adb shell dumpsys battery

* This shows battery statistics and might help check how much power is being consumed while the device is in standby.

### ****6. Monitor Device’s Network and Power State****

You can track the network connectivity and power usage during standby mode with the following:

**adb shell dumpsys connectivity**

**adb shell dumpsys power**

* These commands show network state and power usage, which is useful for identifying whether certain background activities (like syncing or network access) occur during standby.

### ****7. Simulate No Activity (Idle)****

To simulate no user activity or interaction with the device, you can leave it idle and run the following command to ensure the screen stays off or the device enters standby:

adb shell input keyevent 26

adb shell svc power stayonfalse

* This turns off the screen and disables the stay awake mode, ensuring the device remains in idle or standby state.

### ****8. Perform Wake-Lock Testing****

To test how the device behaves with wake-locks or when certain apps are keeping the device awake, you can trigger wake-locks using the following commands:

adb shell am broadcast -a com.android.intent.action.WAKE\_LOCK

* This simulates an app or process that prevents the device from going to sleep (useful for testing how certain apps affect standby).

### ****9. Monitor Wake and Sleep Transitions****

Use the following command to track when the device wakes up or goes to sleep:

adb logcat -v time | grep -i"PowerManager"

* This will show the device’s wake and sleep transitions, useful for debugging how background apps or services handle standby and sleep.

### ****10. Simulate Doze Mode (Power Saving Mode)****

Android has a **Doze Mode**, which reduces battery consumption when the device is not in use. You can simulate this by putting the device into Doze mode using:

adb shell dumpsysdeviceidle force-idle

* This command forces the device into Doze mode, simulating long periods of inactivity. You can also check whether your app or services are being restricted in Doze mode.

To exit Doze mode:

**adb shell dumpsysdeviceidle unforce**

## Field Testing

1. Testing an application in the Real Env ( not undr dev testing env it is about production env)
2. Make sure your application will work in every circumstances.
3. Device testing : temp , speed , water proof.

## UI Testing

1. To attract user with different UI strategies

## Uninstallation Testing

1. Check if the application is uninstalling or not along with data realetd to that applciaiotn.
2. Find app package to uninstall
   * 1. How? Open appliaton in ur phone-
     2. Commond:
        1. >adb shell dumpsys window | find “mCurrentFocus”
        2. Package and activity
3. To uninstall enter cmd
   * 1. >adb uninstall “io.appium.android.apis”.

**1. Cold Start**

A cold start happens when the app is launched from scratch—i.e., it is not already running in memory. This requires the app to load completely, including initializing resources and dependencies.

**Steps:**

1. **Close the app completely:**

bash

Copy code

adb shell am force-stop <package\_name>

1. **Start the app:**

bash

Copy code

adb shell am start -W <package\_name>/<activity\_name>

1. **Analyze launch time:**  
   The -W flag captures launch time details. Check the output for metrics like TotalTime, WaitTime, and Displayed.

**2. Warm Start**

A warm start occurs when the app is in the background but not actively visible. The app is still in memory, so launching it doesn’t require full initialization.

**Steps:**

1. **Send the app to the background:**

bash

Copy code

adb shell input keyevent KEYCODE\_HOME

1. **Bring the app back to the foreground:**

bash

Copy code

adb shell am start -W <package\_name>/<activity\_name>

1. **Review the launch time:**  
   Similar to cold start, the -W flag will provide performance data.

**3. Hot Start**

A hot start occurs when the app is already running in the foreground or paused state. No significant resource loading is needed as the app resumes quickly.

**Steps:**

1. **Ensure the app is running in the foreground.**
2. **Simulate a pause and resume:**

bash

Copy code

adb shell input keyevent KEYCODE\_APP\_SWITCH

adb shell input keyevent KEYCODE\_APP\_SWITCH

1. **Verify performance:**  
   Typically, hot starts are nearly instantaneous. Measure response time using logs or stopwatch.

# Capabilities

1. "Capabilities" is the name given to the set of parameters used to start an Appium session. The information in the set is used to describe what sort of "capabilities" you want your session to have, for example, a certain mobile operating system or a certain version of a device.
2. When you start your Appium session, your Appium client will include the set of capabilities you've defined as an object in the JSON-formatted body of the request.
3. Capabilities are represented as key-value pairs, with values allowed to be any valid JSON type, including other objects. Appium will then examine the capabilities and make sure that it can satisfy them before proceeding to start the session and return an ID representing the session to your client library.

|  |  |  |
| --- | --- | --- |
| **Capability** | **Type** | **Description** |
| platformName | string | The type of platform hosting the app or browser |
| appium:automationName | string | The name of the Appium driver to use |
| browserName | string | The name of the browser to launch and automate, if the driver supports web browsers as a special case |
| appium:app | string | The path to an installable application |
| appium:deviceName | string | The name of a particular device to automate, e.g., iPhone 14 (currently only actually useful for specifying iOS simulators, since in other situations it's typically recommended to use a specific device id via the appium:udid capability). |
| appium:platformVersion | string | The version of a platform, e.g., for iOS, 16.0 |
| appium:newCommandTimeout | number | The number of seconds the Appium server should wait for clients to send commands before deciding that the client has gone away and the session should shut down |
| appium:noReset | boolean | If true, instruct an Appium driver to avoid its usual reset logic during session start and cleanup (default false) |
| appium:fullReset | boolean | If true, instruct an Appium driver to augment its usual reset logic with additional steps to ensure maximum environmental reproducibility (default false) |
| appium:eventTimings | boolean | If true, instruct an Appium driver to collect [Event Timings](https://appium.io/docs/en/2.0/guides/event-timing/) (default false) |
| appium:printPageSourceOnFindFailure | boolean | If true, collect the page source and print it to the Appium log whenever a request to find an element fails (default false) |

**AppiumBy Class**

|  |  |
| --- | --- |
| **Modifier and Type** | **Class** |
| static class | [**AppiumBy.ByAccessibilityId**](https://javadoc.io/static/io.appium/java-client/8.0.0/io/appium/java_client/AppiumBy.ByAccessibilityId.html) |
| static class | [**AppiumBy.ByAndroidDataMatcher**](https://javadoc.io/static/io.appium/java-client/8.0.0/io/appium/java_client/AppiumBy.ByAndroidDataMatcher.html) |
| static class | [**AppiumBy.ByAndroidUIAutomator**](https://javadoc.io/static/io.appium/java-client/8.0.0/io/appium/java_client/AppiumBy.ByAndroidUIAutomator.html) |
| static class | [**AppiumBy.ByAndroidViewMatcher**](https://javadoc.io/static/io.appium/java-client/8.0.0/io/appium/java_client/AppiumBy.ByAndroidViewMatcher.html) |
| static class | [**AppiumBy.ByAndroidViewTag**](https://javadoc.io/static/io.appium/java-client/8.0.0/io/appium/java_client/AppiumBy.ByAndroidViewTag.html) |
| static class | [**AppiumBy.ByClassName**](https://javadoc.io/static/io.appium/java-client/8.0.0/io/appium/java_client/AppiumBy.ByClassName.html) |
| static class | [**AppiumBy.ByCustom**](https://javadoc.io/static/io.appium/java-client/8.0.0/io/appium/java_client/AppiumBy.ByCustom.html) |
| static class | [**AppiumBy.ById**](https://javadoc.io/static/io.appium/java-client/8.0.0/io/appium/java_client/AppiumBy.ById.html) |
| static class | [**AppiumBy.ByImage**](https://javadoc.io/static/io.appium/java-client/8.0.0/io/appium/java_client/AppiumBy.ByImage.html) |
| static class | [**AppiumBy.ByIosClassChain**](https://javadoc.io/static/io.appium/java-client/8.0.0/io/appium/java_client/AppiumBy.ByIosClassChain.html) |
| static class | [**AppiumBy.ByIosNsPredicate**](https://javadoc.io/static/io.appium/java-client/8.0.0/io/appium/java_client/AppiumBy.ByIosNsPredicate.html) |
| static class | [**AppiumBy.ByName**](https://javadoc.io/static/io.appium/java-client/8.0.0/io/appium/java_client/AppiumBy.ByName.html) |

### ****Key Differences****:

|  |  |  |
| --- | --- | --- |
| **Feature** | **WebDriver** | **AndroidDriver** |
| **Purpose** | Automates web applications in browsers | Automates native, hybrid, and web apps on Android devices |
| **Platform** | Web (desktop/mobile browsers) | Android (native apps, hybrid apps, mobile browsers) |
| **Library** | Selenium | Appium |
| **Special Methods** | Standard Selenium methods | Android-specific methods (e.g., startActivity()) |
| **Locator Support** | Standard web locators (CSS, XPath) | Appium-specific locators (accessibility ID, UIAutomator) |
| **Device/Browser** | Desktop/mobile browsers | Android devices/emulators |

# Install Appium From the Command Line

Download and install Appium globally with npm by executing the following command in your terminal

npm install -g appium@2.5.4

Next, verify the installed version of Appium by running the following command in your terminal:

appium --version

To get a list of all the available drivers for your OS, run the below command:

appium driver list

# Why to Use Appium Framework?

Mobile testing tools are responsible for ensuring the quantity and functionality of mobile apps to enhance user experience. The tools identify and address the potential issues before app development as these tools minimize the bugs and enhance the overall reliability. Some of the mobile testing tools are Kobiton, [Robotium](https://www.geeksforgeeks.org/appium-vs-robotium/" \t "_blank), Test Complete, and more, but, Appium is preferred over the other testing tools:

* Appium's support for multiple languages enhances scalability and eliminates the need to set up multiple platforms during integration, thereby reducing costs.
* Appium's open-source nature encourages testing on simulators, emulators, and real devices, providing cost-effective testing options.
* Appium offers extensions to work with both native and hybrid mobile applications, ensuring a seamless testing experience without the need for code changes.
* Appium allows users to access back-end APIs and databases directly from their test code, facilitating comprehensive testing practices.
* Users can leverage their preferred testing practices, tools, and frameworks with Appium, ensuring flexibility and adaptability in their testing processes.
* **Difference between Appium and Selendroid**
* Here's the comparison between Appium and Selendroid
* .

# Limitation of Appium

Appium is a popular testing tool primarily designed for testing mobile applications across various operating systems, including Android and iOS. It utilizes a WebDriver and stands out as an open-source automation tool that provides support for multiple programming languages compatible with WebDriver, such as Java, Objective C, JavaScript, Python, and Ruby. It functions as a versatile cross-platform testing framework, allowing testers to craft test scripts that can operate seamlessly across different platforms like iOS, Windows, and Android.

**Limitations :**

**1) Complexity of Appium Setup:** Appium's setup and configuration can be daunting, especially for beginners. Installing dependencies, configuring drivers, and setting up devices can be complex and time-consuming.

***Alternative****: To simplify the setup process, consider using Appium Desktop or Appium Studio, which offer user-friendly interfaces for configuring Appium servers.*

**2) Limitations in Performance Testing:** Appium primarily focuses on functionality testing and user interface automation, offering limited support for performance testing aspects like load testing and stress testing.

***Alternative****: For comprehensive performance testing, you may need to integrate Appium with specialized performance testing tools such as JMeter or Gatling.*

**3) Limited Desktop Testing:** Appium primarily targets mobile app testing, making it less suitable for automating desktop applications. This can necessitate the use of separate tools for desktop application testing.

***Alternative****: For desktop testing, explore other automation tools like Selenium.*

**4) Heavy Maintenance:** Maintaining scripts written in Appium is essential, and it becomes more challenging for larger projects or enterprise-level automation where scripts need continuous updates, resulting in increased costs.

***Alternative****: Implement strong version control practices and consider using a test management tool to streamline script maintenance.*

**5) Limited Support for Accessibility Testing:** Appium may not provide the same level of support for accessibility testing, especially in features like screen reader testing, as dedicated accessibility testing tools.

***Alternative****: For thorough accessibility testing, consider using specialized accessibility testing tools in conjunction with Appium.*

**6) One-on-One Support / Community Support:** Appium lacks one-on-one support, and quick solutions for issues may not always be readily available. Instead, it relies on community forums for support.

***Alternative****: Engage with the Appium community forums to seek answers and solutions to your queries and issues.*

**7) Supports Only Newer Devices:** Appium supports Android versions from 4.2 onwards, limiting its compatibility with older Android versions.

***Alternative****: For older Android versions, consider alternative testing tools or adapt your testing strategy accordingly.*

**8) UI Locator Challenges:** Identifying and interacting with UI elements in dynamic or frequently changing environments can be challenging, leading to unreliable tests.

***Alternative****: Employ robust and dynamic element locators, and ensure that your tests are resilient to UI changes.*

**9) Limited Support for Hybrid App Testing:** Appium has limitations in handling transitions between native and web applications in hybrid apps.

***Alternative****: Depending on your use case, consider using a combination of Appium and other tools that specialize in web application testing.*

**10) Language Proficiency:** Automating tests in Appium requires knowledge of a programming language, which can be a barrier for non-technical users.

# Appium UiAutomator2 Driver

Appium UiAutomator2 Driver is a test automation framework for Android devices. Appium UiAutomator2 Driver automates native, hybrid and mobile web apps, tested on emulators and real devices. Appium UiAutomator2 Driver is part of the [Appium](https://github.com/appium/appium) mobile test automation tool. The driver operates in scope of [W3C WebDriver protocol](https://www.w3.org/TR/webdriver/) with several custom extensions to cover operating-system specific scenarios.

UiAutomator2 Driver proxies most of the commands to [UiAutomator2 server](https://github.com/appium/appium-uiautomator2-server), which uses Google's [UiAutomator](https://developer.android.com/training/testing/ui-automator) framework under the hood. Some commands are proxied directly to [appium-adb](https://github.com/appium/appium-adb) and other helpers built on top of Android platform tools.

# Gestures

## 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
* *locator*: The map containing [strategy and selector](https://github.com/appium/appium-uiautomator2-driver/blob/master/README.md#element-location) items to make it possible to click dynamic elements.

**Usage examples**

// Java

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

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

));

## 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
* *locator*: The map containing [strategy and selector](https://github.com/appium/appium-uiautomator2-driver/blob/master/README.md#element-location) items to make it possible to click dynamic elements.

**Usage examples**

// Java

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

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

));

## 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
* *locator*: The map containing [strategy and selector](https://github.com/appium/appium-uiautomator2-driver/blob/master/README.md#element-location) items to make it possible to click dynamic elements.

**Usage examples**

// Java

driver.executeScript("mobile: clickGesture", ImmutableMap.of(

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

));

## 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

**Usage examples**

// Java

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

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

"endX", 100,

"endY", 100

));

## 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

**Usage examples**

// Java

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

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

"direction", "down",

"speed", 500

));

## 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

**Usage examples**

// Java

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

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

"percent", 0.75

));

## 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

**Usage examples**

// Java

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

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

"percent", 0.75

));

## 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

**Usage examples**

// Java

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

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

"direction", "left",

"percent", 0.75

));

## 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

**Usage examples**

// Java

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

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

"direction", "down",

"percent", 1.0

));

**Guide on UiAutomator Locator Types**

UIA2 driver enables elements lookup using [UiSelector](https://developer.android.com/reference/androidx/test/uiautomator/UiSelector). [UiScrollable](https://developer.android.com/reference/androidx/test/uiautomator/UiScrollable) is also supported. Both locator types are supported natively by Google's [UiAutomator](https://developer.android.com/training/testing/other-components/ui-automator) framework for Android. With these locators you could create flexible ways to reference complicated element paths, and their performance is very close to native.

WebElementele =

driver.findElement(AppiumBy.*androidUIAutomator*("new UiScrollable(new UiSelector()).scrollIntoView(text(\"India/English\"));"));

**My First Script**

public class appiumcalc()

{

DesiredCapabilitiesdcap=new DesiredCapabilities();

dcap.setCapability(“platformName”,”android”);

dcap.setCapability(“automationName”,”uiautomator2”);

dcap.setCapability(“deviceName”,”vivo v21e 5G);

dcap.setCapability(“UDID”,””);

dcap.setCapability(“appPackage”,””);

dcap.setCapability(“appActivity”,””);

URL u=new URL(<http://localhost:4723>);

AndroidDriver driver=new AndroidDriver(u,dcap);

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));

}