Appium Testing Tutorial

Introduction

Appium is an open-source framework for automating native, hybrid, and mobile web applications across iOS and Android. It can be complex to set up initially for beginners; hence, the big pain points are mostly related to elements locators-like overusing slow XPath-and handling of synchronization, waits, which is leading to frustrating, flaky tests.

Our comprehensive course is specifically designed to tackle these challenges head-on, giving you a smooth start and the skills to build robust, maintainable mobile test automation frameworks. Ready to overcome the learning curve and master cross-platform testing? View Our **Appium Testing Course Syllabus**.

Why Students or Freshers Learn Appium Testing?

Learning Appium becomes the main priority for freshers and students entering the QA field because of the very high demand for mobile automation.

- **Cross-Platform Expertise:** Appium enables you to maintain a single set of test scripts in any language-Java, Python, etc.-that work seamlessly across iOS and Android on native, hybrid, and web apps, giving you a massive skill advantage.
- **High Market Demand:** With millions of mobile apps in existence, companies around the world are looking for testers who could automate using Appium efficiently-an industry-standard, open-source framework.
- **Career Foundation:** It builds on the core programming and Selenium knowledge one has and develops a high-value specialized skill that offers better job prospects and salary.
- **Real-World Projects:** You learn to test on both real devices and emulators/simulators, preparing you directly for professional development pipelines.

Ready to ace your mobile automation interview? Download our free **Appium Interview Questions and Answers** guide now!

Step-by-Step Appium Testing Tutorial for Beginners

Appium is the open-source solution that enables a single code base to write tests for native, hybrid, and mobile web applications for both iOS and Android. This tutorial will take you through the important steps involved, right from the environment setup to writing your first test script.

Step 1. Prerequisites and Environment Setup

The Appium server is built on Node.js, and it uses platform-specific drivers to interact with devices; therefore, you'll need a few key tools installed.

Important Components for Appium Testing

- **Java Development Kit:** Required to run Java-based test scripts – if using Java, and the Android SDK. **Tip:** Set the JAVA_HOME environment variable.
- **Node.js and npm:** Appium Server is built on Node.js; npm (Node Package Manager) is used to install Appium. **Tip:** Install the official version.
- **Android Studio & SDK:** Required for testing Android: it contains the Android SDK, ADB (Android Debug Bridge), emulators (AVDs), and several tools. **Tip:** Set the ANDROID_HOME environment variable and add platform-tools and tools/bin to your system PATH.
- **Xcode (for iOS only):** Required for iOS testing. Only available on macOS. Provides the iOS Simulators and WebDriverAgent (WDA). **Tip:** Install the App Store, then open a terminal and run xcode-select –install
- **Appium Server:** The central component gets WebDriver commands from your script and translates them into mobile-specific actions. **Tip:** Install using npm: npm install -g appium.
- **Appium Doctor:** A helpful diagnostic tool to check if your environment is set up right. **Tip:** Install using npm: npm install -g appium-doctor and run appium-doctor to verify.

Appium Installation & Verification

Install Appium Server: Run the following command in your terminal/command prompt:

npm install -g appium

Verify Setup with Appium Doctor: Run the following to check for missing dependencies:

appium-doctor

Step 2. Configuration of Target Device/Emulator

Before running a test, Appium needs a device or emulator/simulator that it will execute on.

2.1. Android Setup (Emulator or Real Device)

Emulator (AVD): Use the AVD Manager in Android Studio to create a Virtual Device. Make a note of the AVD Name chosen.

Real Device:

- Enable Developer Options on your Android device – this is usually done by tapping the “Build number” 7 times in Settings > About Phone.
- Enable USB Debugging inside Developer Options.
- Connect your device via USB. Run adb devices in your terminal to confirm that your device is listed.

2.2. iOS Setup (Simulator or Real Device – macOS only)

Simulator: Create and manage simulators with Xcode.

Real Device: An Apple Developer Account is required, and proper Provisioning Profiles are necessary to sign and install the WebDriverAgent (WDA), which Appium uses for automation.

Step 3. Start the Appium Server

The Appium server must be running to receive commands from your test script.

Launch the Server: In your terminal, simply execute:

appium

You should see output that shows the server has started and is listening on default host (127.0.0.1) and port (4723).

Note: For beginners, you can also use Appium Desktop (or Appium Server GUI) which provides a simple graphical interface for starting the server and viewing logs.

Step 4. Writing Your First Appium Test

For this example, we'll use Java with the Appium Java Client along with TestNG; this is quite a common setup in professional environments.

4.1. Project Setup (Maven Example)

Create a new Maven project and in your pom.xml, add the following dependencies:

```
<dependencies>

    <dependency>

        <groupId>io.appium</groupId>

        <artifactId>java-client</artifactId>

        <version>8.6.0</version> </dependency>

    <dependency>

        <groupId>org.testng</groupId>

        <artifactId>testng</artifactId>

        <version>7.8.0</version>

        <scope>test</scope>
```

</dependency>

</dependencies>

4.2. Desired Capabilities: The Setup Instructions

The Desired Capabilities are a set of key-value pairs (sent as a JSON object) which describe to the Appium server what kind of session you want to start: which platform, which device, which app, etc.

For Android, we employ the UiAutomator2Options class – the modern equivalent of DesiredCapabilities in the Java client:

Example Capabilities for Android:

Capability Name	**Value**	**Purpose**
platformName	“Android”	The OS platform.
deviceName	“Pixel_3a_API_30”	The AVD or real device name.
platformVersion	“11.0”	The OS version.
automationName	“UiAutomator2”	The automation engine/driver to use.
app	/path/to/my/app.apk	Absolute path to the app file. (Or use appPackage and appActivity if already installed).

appPackage	“com.android.settings”	The Java package of the app.
appActivity	“.Settings”	The main activity to launch.

4.3. The Java Code

Below is a simple test that launches the Android Settings application.

```
import io.appium.java_client.android.AndroidDriver;

import io.appium.java_client.android.options.UiAutomator2Options;

import org.openqa.selenium.remote.DesiredCapabilities;

import org.testng.annotations.Test;

import java.net.MalformedURLException;

import java.net.URL;

public class BasicAndroidTest {

    @Test

    public void launchSettingsAppTest() throws MalformedURLException {

        // 1. Configure Desired Capabilities (Appium 2.x style using Options)

        UiAutomator2Options options = new UiAutomator2Options()

                .setPlatformName(“Android”)
```

```
        .setDeviceName("emulator-5554") // Replace with your device/emulator name
        .setPlatformVersion("11.0")       // Replace with your OS version
        .setAutomationName("UiAutomator2")
        // Using the package and activity of the built-in Settings app for this demo
        .setAppPackage("com.android.settings")
        .setAppActivity(".Settings");
// 2. Define the Appium Server URL
URL appiumServerURL = new URL("http://127.0.0.1:4723/wd/hub");
// 3. Initialize the Driver
// The driver starts the automation session based on the options provided.
AndroidDriver driver = new AndroidDriver(appiumServerURL, options);
try {
    // 4. Perform a simple interaction (e.g., waiting 5 seconds)
    System.out.println("App launched successfully! Waiting 5 seconds…");
    Thread.sleep(5000);
    // 5. Verify a simple element is present (a basic assertion)
    // Locating the 'Search' element in the Settings app
```

```
            if (driver.findElementByAccessibilityId("Search").isDisplayed()) {
                System.out.println("Search element found. Test Passed!");
            } else {
                System.out.println("Search element not found. Test Failed!");
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            // 6. Quit the session
            if (driver != null) {
                driver.quit();
                System.out.println("Appium session closed.");
            }
        }
    }
}
```

Step 5. Element Inspection and Locators

One of the most important steps involves identifying appropriate locators for UI elements. You will be using Appium Inspector.

5.1. Using Appium Inspector

- **Install Appium Inspector:** download the standalone application or install via npm (npm install -g appium-inspector).
- **Start a Session:**
    - Open Appium Inspector.
    - Appium Server URL : http://127.0.0.1:4723
    - Now paste the exact Desired Capabilities you used in your Java code. You can paste the JSON.
    - Click Start Session.
- **Inspect Elements:** The inspector will launch your app on the target device/emulator. Clicking or hovering over an element in the screenshot will show you its unique attributes, like resource-id for Android or accessibility-id for both platforms.

5.2. Best Practice Locators

Always in this order of importance:

1. **Accessibility ID:** most reliable and cross-platform friendly. It's used for accessibility tools and is often a unique identifier.
    - **Android:** content-desc
    - **iOS:** accessibility-id
    - **Code Example (Java):** *driver.findElementByAccessibilityId("Search");*
2. **ID/resource ID:** Unique identifiers, very fast, but platform-specific.
    - **Android:** resource-id
    - **Code Example (Java):**
        - *driver.findElementById("com.android.settings:id/search_action_bar");*

3. **Class Name:** The type of UI element – for example, android.widget.TextView, XCUIElementTypeButton. Useful but often non-unique.
    - **Code Example (Java):**
        - driver.findElementsByClassName(“android.widget.TextView”);
4. **XPath:** Ever powerful yet painfully slow, brittle being subject to breaking even on minor UI changes; only using relative XPaths as an absolutely last resort.

Step 6. Advanced Topics: Gestures and Waits

6.1. Handling Gestures (Swipe Example)

Mobile apps require the use of gestures like swiping or scrolling. The Appium Java Client supports both PointerInput and W3C actions for complex gestures.

```
import org.openqa.selenium.Dimension;

import org.openqa.selenium.Point;

import org.openqa.selenium.interactions.Pause;

import org.openqa.selenium.interactions.PointerInput;

import org.openqa.selenium.interactions.Sequence;

import java.time.Duration;

import java.util.Collections;

public void swipeUp(AndroidDriver driver) {

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

    int startX = size.getWidth() / 2;
```

```
    int startY = (int) (size.getHeight() * 0.8); // 80% down

    int endY = (int) (size.getHeight() * 0.2);   // 20% down (swipe up)

    PointerInput finger = new PointerInput(PointerInput.Kind.TOUCH, “finger”);

    Sequence swipe = new Sequence(finger, 1);

    // Move finger to start point

    swipe.addAction(finger.createPointerMove(Duration.ZERO,
PointerInput.Origin.viewport(), new Point(startX, startY)));

    // Press down


swipe.addAction(finger.createPointerDown(PointerInput.MouseButton.LEFT.asKey()));

    // Pause for a short duration

    swipe.addAction(new Pause(finger, Duration.ofMillis(50)));

    // Move finger to end point

    swipe.addAction(finger.createPointerMove(Duration.ofMillis(500),
PointerInput.Origin.viewport(), new Point(startX, endY)));

    // Lift finger

    swipe.addAction(finger.createPointerUp(PointerInput.MouseButton.LEFT.asKey()));

    // Perform the action
```

```
    driver.perform(Collections.singletonList(swipe));
}
```

6.2. Explicit Waits Implementation

Do not use *Thread.sleep()*. Explicit Waits should be used while waiting for a condition. That will make the tests stable and faster.

```
import org.openqa.selenium.support.ui.WebDriverWait;

import org.openqa.selenium.support.ui.ExpectedConditions;

import org.openqa.selenium.By;

import io.appium.java_client.AppiumBy;

import java.time.Duration;

// … inside your test method

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));

// Wait until an element is visible before interacting with it

wait.until(ExpectedConditions.visibilityOfElementLocated(

    AppiumBy.accessibilityId("your_element_id")

)).click();
```

Ready to deep-dive into the complex issues of mobile automation and learn how to write truly robust, production-ready tests? Learn how to overcome advanced issues with this guide on **Advanced Appium Testing Challenges and Solutions**.

Real Time Examples for Appium Testing Tutorial for Learners

Appium is strong when performing tests for real user interactions running on mobile devices. The following are common scenarios in which automation using Appium is ideal; this helps learners understand practical usage:

User Registration and Login Flow:

- Automate the input of user data: email and password into text fields.
- Handling the soft keyboard during input.
- Tapping on the “Register” or “Login” button and checking whether this successfully leads to the home screen.
- Test cases for failures, such as invalid credentials or missing fields.

E-commerce Product Search and Purchase:

- Interacting with search bars using the send_keys command.
- Scrolling and swiping to browse product lists and categories.
- Tapping an image/link to navigate to the Product Detail Page-PDP.
- Add to Cart functionality testing & verify if cart items count updates or not.

Location and Camera Permissions:

- Handle system alerts, such as “Allow app to access location?”, using the driver.switchTo().alert().accept() method.
- Testing that an app feature such as map view or photo upload does not work until permission is granted.

Cross-Platform UI Verification:

- Using accessibility IDs to ensure the same element, like a “Submit” button, is clickable and positioned in the exact, identical position on both iOS and Android devices for consistency in user experience.

Are you ready to apply these concepts in practical applications? Check out our list of **Appium Testing Project Ideas** and build your portfolio!

FAQs About Appium Testing Tutorial for Beginners

1.How to start with Appium testing?

Set up your environment by installing Node.js, Appium Server via npm, and platform SDKs such as the Android SDK or Xcode. Then, verify the setup with Appium Doctor, configure the Desired Capabilities for your target device and app, and write a simple test script in Java or Python.

2.Is Appium easy to learn?

Appium has a steeper initial learning curve because of complex environment setup like SDKs and environment variables, and mobile-specific challenges such as handling gestures and permissions. At the same time, if you already know Selenium WebDriver, the core scripting logic for element interaction is fairly familiar and relatively easy to adapt.

3.Which IDE is best for Appium?

The best IDE mostly depends on your programming language. For Java-based Appium projects, which is very common, use IntelliJ IDEA or Eclipse. If you're into Python, PyCharm is the best option. All modern IDEs do support the client library and test frameworks like TestNG/JUnit.

4.Is Appium better than Selenium?

Appium is not "better" than Selenium; it's an extension on top of the same WebDriver protocol. Selenium is for web browsers, while Appium is specifically designed for mobile applications-both native, hybrid, and mobile web-on iOS and Android. They serve different testing targets by using a similar API structure.

5.What skills are needed for Appium?

A strong foundation in a programming language like Java, Python, or JavaScript; an understanding of mobile testing concepts; and most importantly, skills with the Appium

framework. It's also good to know about mobile operating systems, locators (like Accessibility ID), test frameworks (e.g., TestNG), and CI/CD tools.

6.Can I learn testing 3 months?

Yes, you will be able to learn the basics of Appium testing in 3 months; especially if you commit a good amount of time to hands-on projects. You can cover setup, scripting, element identification, basic gestures, and integration with a testing framework like TestNG within this timeline. Of course, with constant practice.

7.Is Appium Java or Python?

Appium itself is a server built upon Node.js. However, you write your test scripts using Appium Client libraries, which are available in multiple languages, including Java, Python, JavaScript, and even C#. Among these, both Java and Python are the highly popular choices for Appium test automation.

8.What is the salary of automation tester in TCS?

Salaries for Automation Testers in TCS (India) are very role-specific and dependent on experience level and city. Entry-level salaries command ₹4.5 LPA to ₹8 LPA (1-2 years), while experienced professionals can get significantly larger packages with experience exceeding 5+ years, based on their niche automation expertise.

9.What is the salary of Appium tester?

The **average salary for an experienced Appium Automation Tester** usually stays competitive in India, between ₹7 LPA and ₹15 LPA, which depends on experience and company. In fact, many top professionals tend to earn well over that amount with much experience in frameworks and CI/CD.

10.Is Appium a black box testing?

Yes, Appium is essentially a black box test since all it tests the compiled application (.apk or .ipa) via its UI by simulating the user steps and doesn't require knowing the source code of an app or how things are implemented inside it. This approach mimics the end-user experience.

Conclusion

You have now gone through the steps you need to launch your first cross-platform mobile test with Appium. You have understood the main concepts: setting up the environment, defining Desired Capabilities, and using the Appium Inspector for reliable locators.This leads to the need to turn these foundational skills into job-ready expertise by building a robust, professional test automation framework. Master gestures, integrate with CI/CD, and implement advanced problem-solving with our comprehensive **Appium Testing Course in Chennai**. Join today and take your career to the next level!