ACS SOLUTIONS

Mobile Automation Testing using Appium

WHITE PAPER

Table of Contents

Lead Authors
Geetha Pavani A
Sai Pawan L

About ACS Solutions

ACS Solutions is a leading global information technology services and consulting organization that has been serving businesses globally across industries since 1998. A trusted partner to both mid-market and Fortune 500 clients, ACS Solutions has been instrumental in each of their unique digital transformation journeys. Our extensive industry-specific domain expertise and passion for innovation has helped clients envision, build, scale and run their businesses more efficiently, for over two decades.

We have a proven track record of developing large and complex software and technology solutions for Fortune 500 clients such as Microsoft, Blue Cross Blue Shield, Cox Communications, and Novartis. We deliver on our commitments and enable our customers to achieve a digital competitive advantage through flexible and global delivery models, agile methodologies, and expert frameworks. Headquartered in Duluth, GA, and with several locations across North America, Europe, and the Asia-Pacific regions, ACS Solutions specializes in 360-degree digital transformation and IT consulting services.

For more information, please reach us at –
acssolutions@acsicorp.com

Executive Summary

Due to the extensive amount of benefits that the mobile apps bring including easy accessibility, enhanced user engagement and retention, there has been a drastic shift towards the usage of mobile apps. Given the increased use of the mobile apps, testing those apps has become even more challenging in terms of covering an exhaustive list of models. This paper outlines the basic strategies and structure for a successful Mobile Automation Testing using Appium. It highlights the common mistakes that are being made while automating the mobile testing process. It also defines the key best practices to be followed to have an efficient mobile app test automation using Appium.

Introduction

The adoption of mobile phones in the last decade has been staggering to say the least. Today, we have close to 5 billion mobile users and roughly 50% of the web page views across the world are from mobile devices. This astonishing increase in the smartphones usage worldwide shows no decline and if the forecasts are to be believed, the growth is not coming down anytime in the near future. Alongside the smart phones, the market for mobile applications also have grown at an equally rapid pace, if not more. Growth always brings with it the associated challenges. Mobile apps face the challenge of supporting multiple devices, platforms and operating systems while providing a consistent experience to the end users. This not only opens a development challenge but also an equally tough challenge of testing these applications to ensure a flawless end user experience. Given the nature of problem and the number of variables involved, manual testing is an extremely difficult proposition and hence the need for mobile automation testing.

Why Appium is best choice?

Appium became the major player in test automation landscape due to following reasons.

Free of cost: The best of its features are free of cost and open source. Also, it doesn't require any device installation to bridge the interaction between the software and the application under test.

Highly Flexible: It supports multiple scripting languages like Java, Python, C#, Javascript etc. that makes it easy for the user to select a scripting language based on their convenience and work across multiple platforms. Also testing the native apps using Appium doesn't require any SDK or app recompiling. In fact, in most of the cases, it doesn't even require any code change to work on Android and iOS.

Easy to learn: As Appium is built on Selenium, it doesn't require any time for the Selenium engineers to ramp up on the tool. Apparently, Appium is a wrapper that translates Selenium commands into iOS and Android commands to interact with the elements of the application under test. Furthermore, all Selenium functionality is available in Appium.

High Community Support: Another major benefit which Appium brings in is the large community of contributors available on all the major networking portals and are striving to keep the users updated on the latest trends on the tool.

Appium Inspector: This can be used as record and playback option like Selenium IDE. Using this the actions on the native apps can be recorded and converted to a selected scripting language for further optimization and customization. But the Appium Inspector isn't compatible with Microsoft Windows currently.

Cross-Platform Automation: Appium supports cross-platform automation wherein tests can be built in any language for both Android and for iOS. These can be executed without any change in the code. It enables tests to be written and run across multiple devices simultaneously, reducing time to market and man-hours of manual test effort, and increased testing coverage across devices that would otherwise not be tested due to time constraints. It can integrate easily with emulators, simulators and cloud environments which could be useful in bringing down the execution time and improve ROI.

Existing Frameworks: Users are free to use their own test practices or frameworks from the already available lot. The framework also allows automation on native, web and hybrid apps. Testing is possible on a real device, simulators or emulators.

Integration with CI tools: Appium framework can be easily integrated with all the leading CI tools that enable integration with the development release cycles.

Common mistakes

- Not performing PoC on the application before the main automation phase
- Not using best locator strategy that might increase the time taken to identify the element
- In Android application, Appium won't identify the elements if it's not visible on page (those are present in page). So we need to swipe the page and check those elements.
- Missing to set environment variables for ANDROID_HOME and JAVA_HOME after installing Java and Android SDK
- Missing to map xcode path for the correct version of xcode as more than one xcode version is possible on the same machine.
- Skipping ideviceinstaller and iOS-deploy installation on Mac machine to execute scripts on iOS device. If there is below error even ideviceinstaller is installed properly on the machine, it can be resolved by setting the below value in project Run configuration Let me know if you need any further info

  Error: Could not initialize ideviceinstaller; make sure it is installed and works on your system(iOS)

  Name: PATH

  Value : /usr/local/bin:/usr/bin:/usr/sbin:/sbin
- If robot class commands are present in the scripts and are executed on Mac machine, background java process would be started and a Java Cup icon is placed in the Dock, it causes the currently active window to lose focus. To resolve this, please specify the attribute "-Dapple.awt.UIElement=true" for jre
- For iOS, only one device or simulator can be run on a mac machine at a time. Careful management and planning are required while scheduling tests across multiple devices at the same time

Best practices to be followed for mobile app automation with Appium

Proof of Concept

In order to have a successful mobile test automation process, it is highly recommended to have a pilot for all the applications as that would give an insight on the estimates and would uncover the challenges and help in proper planning. The following points should be kept in mind while implementing the POC:

- It should be implemented in critical scenarios to understand the scope as the time taken to automate is different for each app type and OS.
- It should be done on the elements that are unique to each OS. For example: the date and time picker.
- Dry run should also be performed to understand the efforts involved

Efforts Estimation

Efforts estimates should cover the following:

- Environment setup
- Scripting the test cases with the challenges faced during the Proof of Concept
- Effort required to make the script compatible with different Operating Systems and devices
- Time taken for the “Dry Run”
- Estimates should include time for analysing, development and unit testing of each scenario

UI Locator Strategy

A proper strategy should be in place to make Appium identify the element in the minimum time possible. The locator should also work if the hierarchy of the elements gets changed in the UI.

The order of the locators to be considered are as follows:

1. Id(Accessibility ID)
2. Name
3. Value
4. ClassName
5. Xpath

While executing Appium scripts the below statement is displayed in Appium logs which means those are the only valid locators that can be used in recent Appium versions. Even we may see many suggestions in intelligence like By .tagName but they are not supported by latest versions of Appium.

[debug] [BaseDriver] Valid locator strategies for this request: xpath, id, name, class name, -ios predicate string, accessibility id

Consider the xml tag of the element shown below.

<XCUIElementTypeButton name="Drive" label="Drive" value="" dom="" enabled="true" valid="true" visible="true" hint="" path="/0/0/0/1/0/0/1/0" x="2" y="619" width="184" height="48"></XCUIElementTypeButton>

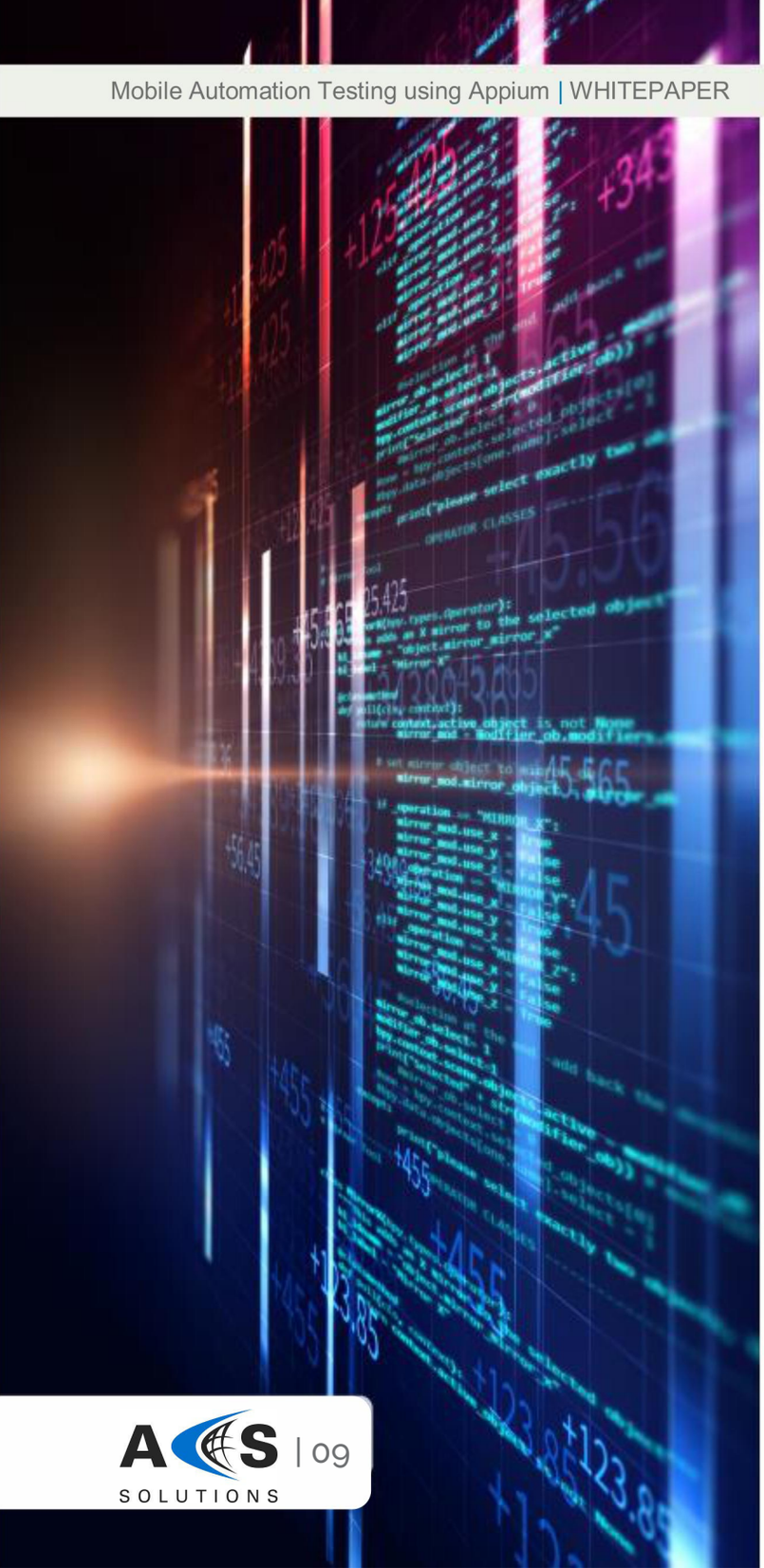
ID(Accessibility ID),Name,Value:

On both platforms Android and iOS, in order to identify a single or multiple elements, considering their accessibility id is usually the best and the preferred way. In iOS there is no concept of ID as we have in android but we can use accessibility id to identify the elements. In the example shown above the name or label is the accessibility id of the element.

```
driver.findElement(By.ID("Drive")).click();
driver.findElementByAccessibilityId("Drive").click();
driver.findElement(By.name("Drive")).click();
driver.findElementByName("Drive").click();
```

All these statements will click on the element with name "Drive". Whenever we want to find the element by name, try to use any one of the first two statements because in Appium 1.6, "name" was replaced with accessibility id.For Android, id is the element's Android:id.

- For iOS strategy is different. Appium will first search for an accessibility id that matches. If there is no match fount, a string match will be attempted on the element labels. Finally, if the id passed in is a localization key, it will search the localized string.
- For iOS it is the full name of a class and will begin with **XCUI,** such as **XCUIElementTypeButton** for a button.
- For Android it is the fully qualified name like **android.widget.EditText** for a text field.
- In case if there is no option other than xpath, then try to use relative xpath instead of absolute xpath because if absolute xpath is used in the locator there may be changes to the parent elements in future which will result in failure of element identification.
- Try to avoid XPath locators unless there are no other alternatives. In general, xpath locators are slower in identifying the elements, than other types of locators like accessibility id, class name. They are slow because xpath location is not natively supported by Apple's XCTest framework.



- Be very specific when locating elements by xpath. Such locators like . Such locators like driver.findElement(By.xpath("//XCUIElementTypeButton[@value='sample']")) is faster than driver.findElement(By.xpath("//*[@value='sample']"))

- Multiple nested findElement calls would be faster than to perform a single call by xpath. **driver.findElement(a).findElement(b)** is faster than **driver.findElement(c)** where a and b are non-xpath locators and c is xpath locator.

Framework

The different factors that decide the effectiveness of the framework are enlisted below:

- **Defined Folder Structure:** Framework should have a well-defined folder structure to easily trace the items
- **Portability:** The framework should work as expected with any folder hierarchy, any drive location and type of application
- **Error Handling and Recovery Management:** The process to be followed when an error or exception occur must be handled in framework rather than in the scripts
- Functional libraries for general use and project specific purpose should be in separate folders for ease of maintenance
- Test data to be maintained on a module basis to update them as and when required without touching the scripts. Also it is needed to ensure the security for the test data
- The framework should be environment independent. By implementing few things that can be configured like browser, OS etc. we can make the framework environment independent.
- The selection of the test scripts to be executed and batch creation should be provided

- There should be an option to select the environment details like OS and device versions along with the environments like emulator or simulator or device or cloud
- There should be a mechanism to schedule the execution of batches
- Test results should be easily understood by any tester with more details on the condition of the application under test and test data used. This will help the tester to reproduce the defect. Capturing the screenshot will also add value to decide if it is a defect or script failure
- There should be a configuration mechanism to email the results to selected recipients.
- Scripting guidelines should be documented
- Naming conventions and standard to be followed should be documented
- Proper trace-ability should be visible to understand the coverage of the automation and execution of the defects
- After the execution, the framework should provide some data for analysis and metrics preparation
- Test framework should be easy to expand and maintain with fewer references to paid tools or other third-party libraries

Above all these, there should be a continuous improvement plan to make the framework more matured. This will lead to a reduction in the manual effort by replacing it with the batch execution.

Cloud integration to speed up execution

As it's costlier to maintain a lab (different devices that cover different device and OS combination) to execute on the different OS and devices, it's easy to get the script executed on the cloud environments like Sauce labs or Test object. This will not only speed up the execution but will also provide a good test coverage.

GRAFT: Ours Ready Automation Framework for Testing

We own a self-crafted integrated accelerator which enables end-to-end automation of Android and iOS applications. It speeds up the automation of functional and regression testing by leveraging the existing automation tools thereby reducing the dependency on highly skilled testing personnel.

The accelerator brings down the initial development effort by 30% by utilizing the predefined methods in the framework. It brings in multiple other benefits including:

- Reduced overall cost due to the open source tools being used
- Increased flexibility of time and resources
- Reduced manual regression test effort by 60% to increase coverage of product areas
- Increased software quality and reliability due to automated testing methods
- Reduced defects and time-to-market
- Reduced automated test development time and faster ROI realization on test automation
- Improved test coverages by executing the scripts on cloud on different environments and platforms including OS versions
- Reduced test automation development phase by over 30%
- Reduced maintenance cost
- Facilitates better communication between various stakeholders and developers, using tables for representing tests and reporting their results
- Reduced dependency on technically skilled resources

Conclusion

In order to achieve success in test automation projects using Appium, one should always consider the below enlisted points

- Resources should be skilled enough to identify the test cases, customize, implement workarounds and troubleshoot
- The framework in place should be highly scalable and easy to maintain
- All the best practices discussed in this paper must be implemented to avoid the common mistakes and achieve success