Getting Started With Appium

WRITTEN BY DAVE HAEFFNER
AUTHOR OF ELEMENTAL SELENIUM

UPDATED BY WIM SELLES
SENIOR SOLUTIONS ARCHITECT, SAUCE LABS

CONTENTS

WHAT IS APPIUM?

Appium is a free and open-source mobile automation framework used for native, hybrid, and mobile web apps. It works on iOS, Android, Mac, and Windows apps using the WebDriver protocol. Appium currently fully supports the W3C (World Wide Web Consortium) specification.

GETTING STARTED

To get up and running on your local machine, you must download an Appium server and client bindings for your preferred programming language. There are Appium language bindings for multiple programming languages. The officially supported ones (alphabetically):

- C# (.NET)
- Java
- JavaScript (Node.js)
- Objective C
- PHP
- Python
- Robot Framework
- Ruby

Before we dive into installing all the Appium dependencies, we are first going to look into the iOS and Android dependencies.

PLATFORM DEPENDENCIES (IOS)

For testing on iOS, you'll need to have a Mac and install:

- Xcode
- Xcode Command Line Tools Package. Install with the `xcode-select --install` command in your terminal once Xcode has been installed.

BASIC SETUP

For basic setup, we recommend the use of Homebrew for installing system dependencies:

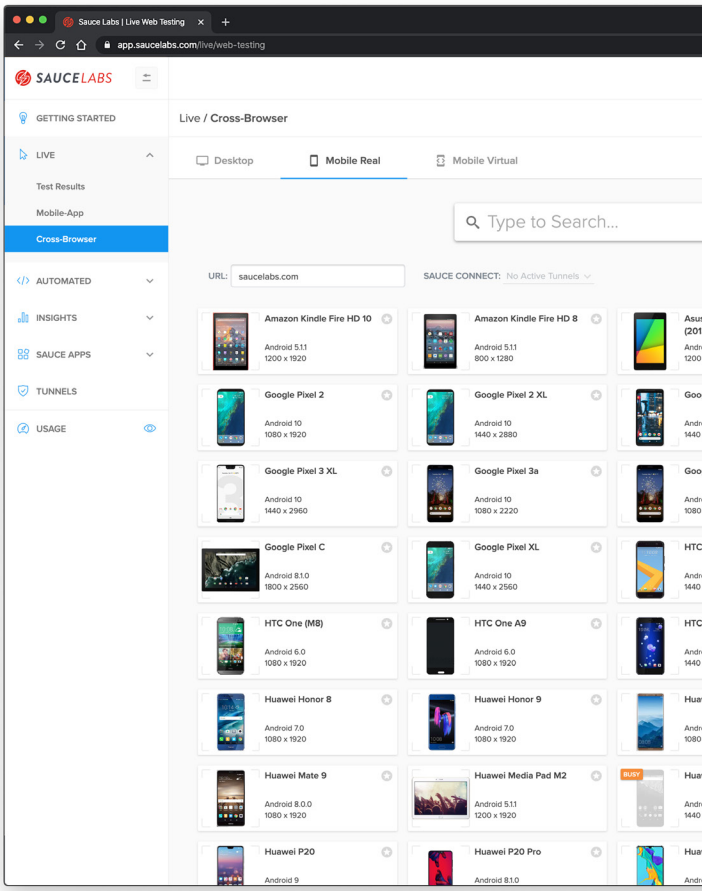
- Ensure that you have Appium's general dependencies (e.g., Node and NPM) installed and configured.
- Install the Carthage dependency manager: `brew install carthage`.

If you don't need to automate real devices, you're done! To automate an app on the simulator, the app capability should be set to an absolute path or URL pointing to your `.app` or `.app.zip` file built for the sim.

REAL DEVICE SETUP

Automating a real device with XCUITest is considerably more complex due to Apple's restrictions around running apps on real devices. Please refer to the XCUITest real device setup doc for instructions.

Your physical devices are sitting in your office, but you're not.



Test remotely from anywhere, on any device, with the Sauce Labs Real Device Cloud.

Start your free trial today

saucelabs.com/sign-up

Email sales@saucelabs.com or call (855) 677-0011 to learn more.

SAUCELABS

Continuous Confidence

Once set up, running a session on a real device is achieved by using the following desired capabilities:

- `app` or `bundleId` — Specifies the application (local path or URL referencing your signed `.ipa` file), or if it is already installed, simply the bundle identifier of the app so that Appium can launch it.
- `udid` — The specific id of the device to test on. This can also be set to auto if there is only a single device, in which case Appium will determine the device ID and use it.

Note: *For additional information on system setup requirements (since your needs might be different), be sure to check out the Appium documentation.*

PLATFORM DEPENDENCIES (ANDROID)

For testing on Android, you'll need to do the following:

- Install JDK 8 or higher
- Download and install Android Studio

Once done, open Android Studio and go to "Tools > SDK manager" and select the SDKs you want to use. Then, go to "Tools > AVD manager" to create an emulator for your tests to use (for the examples that follow, we'll be using a Google Pixel with Android 8.1).

Note: *For additional information on system setup requirements (since your needs might be different), be sure to check out the Appium documentation.*

APPIUM SERVER

After installing all the iOS and Android platform dependencies, you have two approaches for getting the Appium Server onto your machine. You can use the command-line server available through npm and install it with `npm install -g appium`.

Alternatively, you can use the Appium Desktop app — an open-source app for Mac, Windows, and Linux that gives you the Appium server in a simple and flexible UI (along with some extra functionality). You can download and install the latest version here.

After you installed Appium, you need to verify if your environment is set up to run Appium. This can be done with appium-doctor.

APPIUM-DOCTOR

Appium uses more dependencies than just the Appium server. To check if all dependencies are installed, you can use appium-doctor, a small npm package that can diagnose and fix common Node, iOS, and Android configuration issues before starting Appium. appium-doctor can be installed with `npm install -g appium-doctor`.

RUNNING APPIUM-DOCTOR

appium-doctor can diagnose:

- iOS and Android setup together by running `appium-doctor`
- iOS only by running `appium-doctor --ios`
- Android only by running `appium-doctor --android`

For example, when you run `appium-doctor` (for iOS and Android diagnostics), you may get the following output:

```
appium-doctor
info AppiumDoctor Appium Doctor v.1.9.0
info AppiumDoctor ### Diagnostic for necessary
dependencies
starting ###
info AppiumDoctor ✔ The Node.js binary was found at:
/Users/exampleUser/.nvm/versions/node/v10.15.2/bin/node
info AppiumDoctor ✔ Node version is 10.15.2
info AppiumDoctor ✔ Xcode is installed at:
/Applications/Xcode.app/Contents/Developer
info AppiumDoctor ✔ Xcode Command Line Tools are
installed in:
/Applications/Xcode.app/Contents/Developer
info AppiumDoctor ✔ DevToolsSecurity is enabled.
info AppiumDoctor ✔ The Authorization DB is set up
properly.
info AppiumDoctor ✔ Carthage was found at: /usr/local/
bin/carthage
info AppiumDoctor ✔ HOME is set to: /Users/wimselles
info AppiumDoctor ✔ ANDROID_HOME is set to:
/Users/exampleUser/Library/Android/sdk
info AppiumDoctor ✔ JAVA_HOME is set to:
/Library/Java/JavaVirtualMachines/jdk1.8.0_191.jdk/
Contents/Home
info AppiumDoctor ✔ adb exists at:
/Users/exampleUser/Library/Android/sdk/platform-tools/
adb
info AppiumDoctor ✔ android exists at:
/Users/exampleUser/Library/Android/sdk/tools/android
info AppiumDoctor ✔ emulator exists at:
/Users/exampleUser/Library/Android/sdk/tools/emulator
info AppiumDoctor ✔ Bin directory of $JAVA_HOME is set
info AppiumDoctor ### Diagnostic for necessary
dependencies
completed, no fix needed. ###
info AppiumDoctor
info AppiumDoctor Bye! Run appium-doctor again when all
manual fixes
have been applied!
info AppiumDoctor
```

If there are any issues that can automatically be fixed, appium-doctor will do that for you. Otherwise, you will get a list of things that you need to fix.

Run `appium-doctor` again to see if all of the fixes were applied in the correct way. When everything is green, you are ready to start testing with Appium.

Note: *The remaining examples will be shown using WebdriverIO, a test framework for Node.js.*

SAMPLE APPS

Don't have a test app? Don't sweat it. There are pre-compiled test apps available to kick the tires with. You can grab an iOS or an Android app here.

Just make sure to put your test app in a known location because you'll need to reference the path to it next.

WEBDRIVERIO

Before you start to automate with WebdriverIO and Appium, you first need to install WebdriverIO. The documentation is descriptive enough to help you install WebdriverIO.

The advice is to start with the setup for the testrunner. The advantage of the testrunner is that it will be an orchestrator for you to start one or multiple drivers at once, by only providing an object of capabilities. This will be handled in the following part.

APP CONFIGURATION

When it comes to configuring your app to run on Appium, there are a lot of similarities to Selenium — namely, the use of capabilities. You can specify the necessary configurations of your app by providing a capabilities array with an object per capability.

Here's what the object looks like for the iOS test app running on an iPhone simulator:

```
capabilities: [
   {
       // The defaults you need to have in your
       // config
       deviceName: 'iPhone X',
       platformName: 'iOS',
       platformVersion: '12.1',
       orientation: 'PORTRAIT',
       // The path to the app
       app: './your/path/to/iOS-Simulator-
NativeDemoApp-0.2.1.app.zip',
       // Read the reset strategies very well, they
       // differ per platform, see
       // http://appium.io/docs/en/writing-running-
       // appium/other/reset-strategies/
       noReset: true,
   },
]
```

And here's what an object looks like for Android on a local emulator:

```
capabilities: [
   {
       // The defaults you need to have in your
       // config
       automationName: 'UiAutomator2',
       deviceName: 'Pixel_8.1', // The name you gave
       // it in Android Studio
       platformName: 'Android',
       platformVersion: '8.1',
       orientation: 'PORTRAIT',
       app: './your/path/to/Android-NativeDemoApp-
0.2.1.apk',
       // Read the reset strategies very well, they
       // differ per platform, see
       // http://appium.io/docs/en/writing-running-
       // appium/other/reset-strategies/
       noReset: true,
   },
]
```

Note: *See a full list of available capabilities here.*

INTERROGATING YOUR APP

Writing automated scripts to drive an app in Appium is very similar to how it's done in Selenium. You first need to choose a ***locator*** strategy. A locator is how you want to find an element. Then you have the ***selector***, which finds an element based on the provided search criteria. The table below includes all available locator strategies for Selenium/Appium:

LOCATOR STRATEGY	FROM	SUPPORT IN APPIUM
`class name`	Selenium	Yes
`id`	Selenium	Yes
`name`	Selenium	Yes
`xpath`	Selenium	Yes
`accessibility id`	Appium	Yes
`-ios predicate string`	Appium	Yes
`-ios class chain`	Appium	Yes
`-android uiautomator`	Appium	Yes
`-ios uiautomation`	Appium	Deprecated
`css selector`	Selenium	No/Yes*
`link text`	Selenium	No/Yes*
`partial link text`	Selenium	No/Yes*
`tag name`	Selenium	No/Yes*

*This is a **No** when a native app is automated, but it is a **Yes** if the app is a hybrid or a web app.*

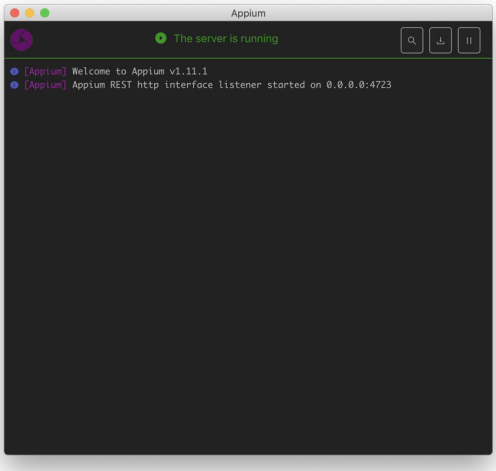
Depending on the app that needs to be automated, there are multiple ways to locate elements. When you are automating a web app, you can use the default tools to locate elements like, for example, Chrome Developer Tools.

Note: *Because a native app is used as an example, we are going to use Appium Desktop to locate the elements.*

USING THE APPIUM DESKTOP APP

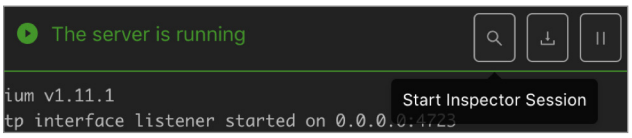
When you download and open Appium Desktop, you will see the following screen:

When you press the Start Server button, the Appium Desktop server will start and a new screen will be shown:

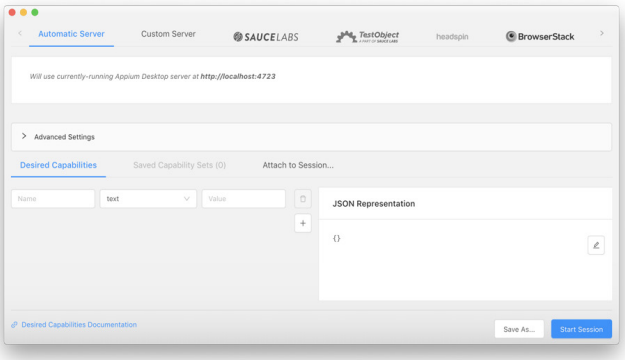


Note: *If you have Appium running in a terminal window, you'll need to kill it by issuing a CTRL+C command.*

Click on the magnifying glass to start the inspector session:



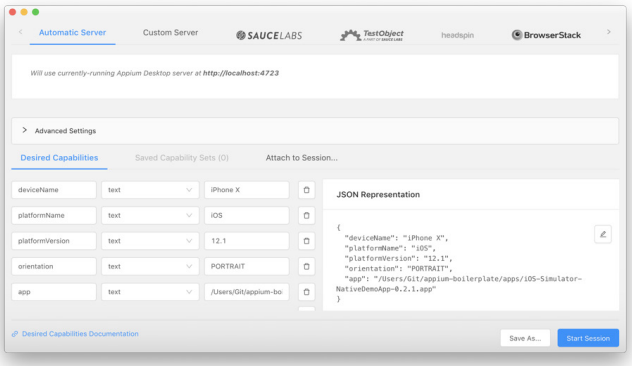
This will start the New Session window:



The New Session window allows you to construct a set of Appium-desired capabilities used to launch an Appium session. You can launch a session against the currently running Appium Desktop server (which is the default), or you can launch a session against a variety of other endpoints.

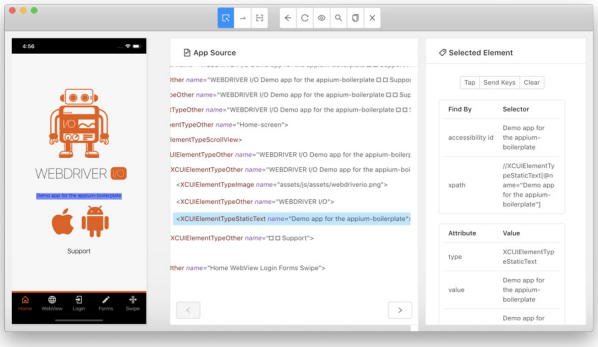
We're going to focus on the automatic server, using the iOS simulator as an example. Android will only differ in capabilities (see above).

The capabilities used for WebdriverIO can also be added here:



Note: *You can save your configuration by clicking the Save As button next to Start Session and giving it a helpful name. Then you can easily refer to this configuration later.*

Press Start Session to start the Appium Desktop Inspector. The three-pane inspector window below shows a screenshot of the app on the left, the underlying UI hierarchy of the app in the center, and details about the element you are attempting to interact with on the right.



In the left pane, you can click on an element you'd like to interact with. When you do, the middle pane will update the source code. The right pane will then:

- Offer actions you can take against it (e.g., Tap or Send Keys).
- Show you one or multiple locator strategies with their selector.
- Show details about the element (attribute values like type, value, visible, etc.).

COMMANDS AND OPERATIONS

The most common operations you'll end up doing in Appium are finding an element — or a set of elements — and performing actions with those elements (e.g., tap, type text, swipe, etc.). You can also ask questions about the elements (e.g., Is it displayed? Is it enabled?), pull information out of the element (e.g., the text of an element or the text of a specific attribute within an element), or perform additional gestures.

FINDING AN ELEMENT

```
// The $ is a shorthand for browser.
findElement('locator', 'selector')
// Find 1 element
const element = $('locator');

// The $$ is a shorthand for browser.
findElements('locator', 'selector')
// Find multiple elements with the same locator
const elements = $$('locator');
```

WORK WITH A FOUND ELEMENT

```
// Chain actions together
$('locator').click();
// Store the element and then click it
const element = $('locator');
element.click();
```

CHAIN ELEMENTS

```
// Add a value to the child within the parent
$('parent').$('child').setValue('type some text');
```

ASK A QUESTION

```
// Check if the element is displayed
$('locator').isDisplayed();
// Check if the element is enabled
$('locator').isEnabled();
```

RETRIEVE INFORMATION

```
// Get the text of an element
$('locator').getText();
// Get the attribute value
$('locator').getAttribute('type');
```

GESTURES

```
it('should do a touch gesture', () => {
    const screen = $('//UITextbox');

    // simple touch action on element
    driver.touchAction({
        action: 'tap',
        element: screen,
    });

    // simple touch action x y variables
    // tap location is 30px right and 20px down
    // relative from the viewport
    driver.touchAction({
        action: 'tap',
        x: 30,
        y: 20,
    });

    // simple touch action x y variables
    // tap location is 30px right and 20px down
    // relative from the center of the element
    driver.touchAction({
        action: 'tap',
        x: 30,
        y: 20,
        element: screen,
    });

    // multi action on an element
    // drag&drop from position 200x200 down 100px on
    // the screen
    driver.touchAction([
        { action: 'press', x: 200, y: 200 },
        { action: 'moveTo', x: 200, y: 300 },
        'release',
    ]);
});
```

Note: *See the documentation for a full list of available commands and operations.*

EXAMPLE TESTS

To give you a quick start, WebdriverIO created:

- A demo app for iOS and Android
- A boilerplate on how to quickly get started with WebdriverIO and Appium

The boilerplate will hold test examples for automating:

- A native app
- A webview (loading a website into an app)
- Safari/Chrome browser

And helpers for:

- Pickers
- Alerts
- Gestures
- And many more

APPIUM SERVICE PROVIDERS

Rather than take on the overhead of standing up and maintaining a test infrastructure, you can easily outsource these services to a third-party cloud provider. With Sauce Labs, for instance, you can access real devices, as well as simulators and emulators.

In the aforementioned appium-boilerplate for WebdriverIO, you'll also find a setup to connect to the real device cloud of Sauce Labs. You can also spin up a Sauce Labs session from within the Appium Desktop app's Start New Session menu.

Note: *You'll need an account to use Sauce Labs. Their free trial offers enough to get you started. And if you're signing up because you want to test an open-source project, then check out their Open Sauce account.*

There's also a handy platform configuration generator that will tell you what values to plug into your test. Explore Sauce Labs' documentation portal for more details.

UPDATED BY WIM SELLES,
SENIOR SOLUTIONS ARCHITECT, SAUCE LABS

Wim Selles is a Solution Architect for Sauce Labs based in the Netherlands. During the day, he assists customers with solving automation challenges in their organization. By night, he practices his passion for front-end test automation with Javascript. He likes to create his own Node.js modules to help and support automation engineers and is also a contributor to multiple open source projects that involve testing, such as WebdriverIO, Protractor, ng-Apimock, and many more. Wim also has extensive experience using Appium for automating Hybrid and React Native Apps.

DZone
A DEVADA MEDIA PROPERTY

DZone, a Devada Media Property, is the resource software developers, engineers, and architects turn to time and again to learn new skills, solve software development problems, and share their expertise. Every day, hundreds of tousands of developers come to DZone to read about the latest technologies, methodologies, and best practices. That makes DZone the ideal place for developer marketers to build product and brand awareness and drive sales. DZone clients include some of the most innovative technology and tech-enabled companies in the world including Red Hat, Cloud Elements, Sensu, and Sauce Labs.

Devada, Inc.
600 Park Offices Drive
Suite 150
Research Triangle Park, NC 27709

888.678.0399 919.678.0300