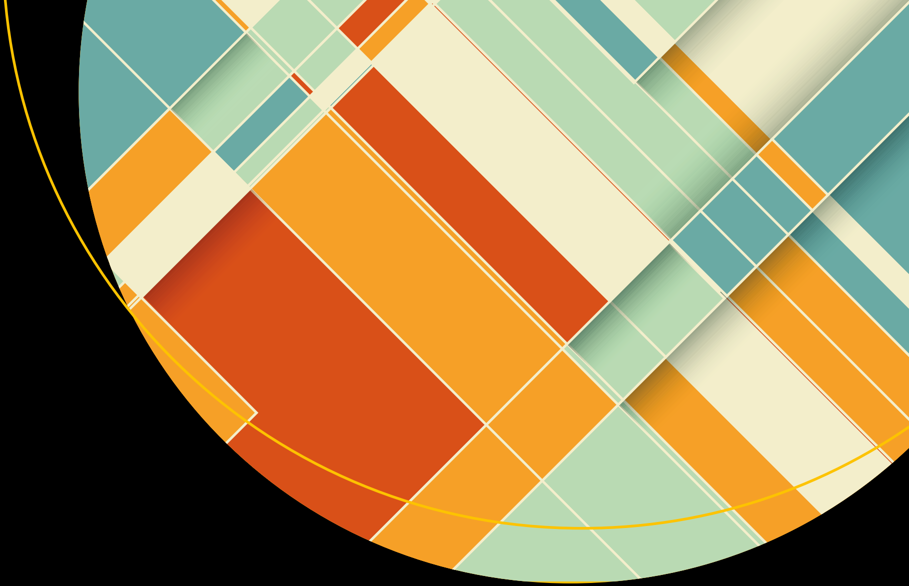


Appium Recipes

Shankar Garg

Apress®

www.allitebooks.com

Appium Recipes

Shankar Garg

Apress®

www.allitebooks.com

Appium Recipes

Shankar Garg
Gurgoan, Haryana
India

ISBN-13 (pbk): 978-1-4842-2417-5 ISBN-13 (electronic): 978-1-4842-2418-2
DOI 10.1007/978-1-4842-2418-2

Library of Congress Control Number: 2016959550

Copyright © 2016 by Shankar Garg

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director: Welmoed Spahr
Lead Editor: James Markham
Technical Reviewer: Unmesh Gundecha
Editorial Board: Steve Anglin, Pramila Balan, Laura Berendson, Aaron Black, Louise Corrigan, Jonathan Gennick, Robert Hutchinson, Celestin Suresh John, Nikhil Karkal, James Markham, Susan McDermott, Matthew Moodie, Natalie Pao, Gwenan Spearing
Coordinating Editor: Sanchita Mandal
Copy Editor: Kim Wimpsett
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text are available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/. Readers can also access source code at SpringerLink in the Supplementary Material section for each chapter.

Printed on acid-free paper

www.allitebooks.com

To my loving wife Shanu Garg.

www.allitebooks.com

Contents at a Glance

About the Author ix

About the Technical Reviewer xi

Introduction xiii

■Chapter 1: Getting Started with Appium... 1

■Chapter 2: Finding Mobile Elements 19

■Chapter 3: Automating Different Apps... 49

■Chapter 4: Automating Mobility 77

■Chapter 5: Creating Automation Frameworks Using Appium..... 101

■Chapter 6: Integrating Appium with Selenium Grid.................... 129

■Chapter 7: Executing Appium with Cloud Test Labs 155

Index.. 179

www.allitebooks.com

Contents

About the Author ix

About the Technical Reviewer xi

Introduction xiii

Chapter 1: Getting Started with Appium 1

1-1. Install Appium via NPM 1

1-2. Run Appium via a GUI App 6

1-3. Understand Capabilities in Appium 11

Chapter 2: Finding Mobile Elements 19

2-1. Traverse with Appium Inspector 19

2-2. Explore UI Automator Viewer 25

2-3. Find Elements by Their Accessibility ID 30

2-4. Find Elements Using iOSUIAutomation 33

2-5. Find Elements Using AndroidUIAutomator 35

2-6. Inspect iOS Mobile Web Elements 38

2-7. Inspect Android Mobile Web Elements 44

Chapter 3: Automating Different Apps 49

3-1. Native Apps 49

3-2. Mobile Web Apps 56

3-3. Hybrid Apps 61

3-4. Real Devices 69

www.allitebooks.com

Chapter 4: Automating Mobility 77

4-1. Tap Mobile Elements 78

4-2. Drag and Drop Elements 82

4-3. Swipe and Scroll 84

4-4. Manage Device Orientation 91

4-5. Install and Uninstall Native Apps 93

4-6. Lock and Unlock Devices 96

4-7. Manage Device Network Settings 98

Chapter 5: Creating Automation Frameworks Using Appium 101

5-1. Create an Automation Framework with Appium, Maven, and TestNG 101

5-2. Create a BDD Framework with Appium, Cucumber, and the Page Object Model 110

5-3. Conduct Continuous Automated Testing with Appium, Git, and Jenkins 122

Chapter 6: Integrating Appium with Selenium Grid 129

6-1. Appium with Selenium Grid for Native App Automation 129

6-2. Appium with Selenium Grid for Mobile Web Automation 141

6-3. Appium with Selenium Grid for Two Android Sessions on the Same Machine 149

Chapter 7: Executing Appium with Cloud Test Labs 155

7-1. Appium on the Sauce Labs Cloud 155

7-2. Appium on the Testdroid Cloud 166

Index 179

www.allitebooks.com

About the Author

Shankar Garg is an Agile enthusiast with expertise in automation testing. He started as a Java developer, but his love for breaking things got him into testing. He has worked on the automation of many projects using web, mobile, and SOA technologies. Right now, he is in love with Cucumber, Selenium, Appium, and Groovy.

He is a Certified Scrum Master (CSM), Certified Tester (ISTQB), and Certified Programmer for Java (SCJP 5.0) and Oracle 9*i* (OCA).

He is the author of *Cucumber Cookbook* (`https://www.packtpub.com/web-development/cucumber-cookbook`). You can find him online at `https://shankargarg.wordpress.com/` and `https://in.linkedin.com/in/shnakeygarg`.

www.allitebooks.com

About the Technical Reviewer

Unmesh Gundecha has a master's degree in software engineering and more than 15 years of experience in agile software development, test automation, and technical QA. He is an agile, open source, and DevOps evangelist with rich experience in a diverse set of tools and technologies. Currently, he is working as an automation architect for a multinational company in Pune, India. Unmesh is the author of *Selenium Testing Tools Cookbook* and *Learning Selenium Testing Tools with Python*.

www.allitebooks.com

Introduction

Appium is an amazing tool that offers a cutting-edge platform for implementing mobile test automation. In fact, Appium's ability to implement test automation for both Android and iOS platforms has made it very popular.

The 30 recipes in this book take you on a learning journey. You will start with basic concepts such as how to start the Appium server, then you will move to advanced concepts such as using `iOSUIAutomator` locator strategies and integrating with Selenium Grid and Jenkins, and finally you will learn to run Appium test cases on cloud labs.

Each chapter has multiple recipes with the first recipe introducing the concepts of that chapter and the later recipes increasing in complexity as you progress with the chapter.

What You Need for This Book

Before starting with Appium, let's make sure you have all the necessary software installed.

The prerequisites for Appium are as follows:

- `Appium.dmg` (Mac)/`Appium.exe` (Windows) (https://bitbucket.org/appium/appium.app/downloads/)
- Node and NPM
    - For iOS (http://blog.teamtreehouse.com/install-node-js-npm-mac)
    - For Windows (www.qoncious.com/questions/install-and-run-nodejs-windows)
- For Android:
    - Android SDK API, version 17 or newer (http://developer.android.com/sdk/index.html)
    - Genymotion Android Emulator (https://www.genymotion.com/)

www.allitebooks.com

- For iOS:
    - MacOS: 10.10 or 10.11.1 recommended
    - Xcode: 6.0 or 7.1.1 recommended (https://developer.apple.com/xcode/download/)
    - Apple Developer Tools (iPhone simulator SDK, command-line tools) and the iOS simulator, version 9.0 or newer
- Java 7 (www.oracle.com/technetwork/java/javase/downloads/index.html)
- Eclipse version 4.4.2 or newer (www.eclipse.org/downloads/)
- Maven (https://maven.apache.org/download.cgi)
- The Eclipse-Maven plug-in (https://marketplace.eclipse.org/content/maven-integration-eclipse-luna-and-newer)
- The Eclipse-TestNG plug-in (https://marketplace.eclipse.org/content/testng-eclipse)
- Jenkins (https://jenkins-ci.org/)
- Git-scm (https://git-scm.com/downloads)

This book was written with the assumption that you already have some experience with mobile testing and mobile automation using Appium. If you're new to mobile automation, you should head over to my blog first to understand the landscape of mobile testing and automation. Here are some pointers:

- Set up the Android software development kit (SDK) and Android emulators.
    - https://shankargarg.wordpress.com/2016/02/25/setup-android-sdk-and-android-emulators/
- Set up the Genymotion Android emulators on Mac OS.
    - https://shankargarg.wordpress.com/2016/02/25/setup-genymotion-android-emulators-on-mac-os/
- Install Xcode, command-line tools, and iOS simulators on Mac.
    - https://shankargarg.wordpress.com/2016/02/29/how-to-install-xcode-command-line-tools-and-ios-simulators-on-mac/
- Create an Appium project by integrating Appium, Eclipse, Maven, and TestNG.
    - https://shankargarg.wordpress.com/2016/02/25/create-an-appium-project-by-integrating-appium-eclipse-maven-testng/

These blogs will help you set up your system for mobile automation and run a basic Appium project.

Code Repository

All the code explained in this book is committed on GitHub at `https://github.com/ShankarGarg/AppiumBook.git`

- `AppiumBookBlog`: Project used in the blogs mentioned earlier to get you started with Appium, Eclipse, TestNG, and Maven and in Chapter 5
- `AppiumRecipesBook`: Project used in Chapters 1 to 7 (except Chapter 5)
- `AppiumCucumberPageObject`: Project used in Chapter 5

What This Book Covers

This book covers the following topics:

- *Chapter 1, "Getting Started with Appium"*: This chapter covers the installation steps for Appium graphical user interface (GUI) app and also Appium via NPM. You will also learn about the important concept of desired capabilities for Appium.
- *Chapter 2, "Finding Mobile Elements"*: This chapter illustrates how to use `UIAutomatorViewer` and Appium Inspector for finding elements for Android and iOS respectively. You will also understand mobile platform native locator strategies such as `AndroidUIAutomator` and `iOSUIAutomator` for Android and iOS, respectively.
- *Chapter 3, "Automating Different Apps"*: This chapter covers how to run different types of apps such as native, mobile web and hybrid apps on both Android and iOS. You will also learn to execute Appium test cases on real devices for Android and iOS.
- *Chapter 4, "Automating Mobility"*: This chapter focuses on Appium's core ability to automate mobile-specific functions such as tapping, dragging and dropping, swiping, scrolling and so on. You will also understand mobile-specific functions such as locking and unlocking, managing network settings, and so on.
- *Chapter 5, "Creating Automation Frameworks Using Appium"*: This chapter covers how to integrate Appium with TestNG and Cucumber to create robust test automation frameworks. You will learn Appium integration with Jenkins and Git to implement continuous integration (CI)/continuous deployment (CD) pipelines.

- *Chapter 6, "Integrating Appium with Selenium Grid"*: This chapter covers Appium integration with Selenium Grid to create an in-premise test infrastructure. You will learn how to execute Appium test cases on Selenium Grid for Android and iOS for single and multiple sessions.
- *Chapter 7, "Executing Appium with Cloud Test Labs"*: This chapter covers Appium integration with the cloud test labs Sauce Labs and Testdroid. You will learn how to execute Appium test cases on cloud test labs that you don't have to maintain.

CHAPTER 1

Getting Started with Appium

In this chapter, you will learn how to do the following:

- Install Appium via Node Package Manager (NPM)
- Run Appium via a graphical user interface (GUI) app
- Understand capabilities in Appium

A few years back, mobile automation was an enigma to everyone, but thanks to Appium, that's not the case anymore. Appium is capable of automating both Android apps and iOS apps, so now there's no need to learn two different tools for two different platforms. Also, since Appium uses the same terminology as Selenium, the learning curve is relatively small for anyone who has used Selenium for web automation. For more information about the basics of Appium, please visit `http://appium.io/slate/en/master/?java#introduction-to-appium`.

This chapter will cover the basics of installing and running an Appium session from GUI and from the command line. Finally, you will create a sample project to run your first Appium script.

1-1. Install Appium via NPM

Problem

The Appium team has been working on rewriting Appium in the latest version of JavaScript, so the team is releasing updated versions of Appium more frequently than before. You get Appium's latest build faster via NPM compared to via the GUI app. So, you need to understand how to run Appium via NPM.

Solution

You need the latest stable version of Node.js and NPM. Please check the introduction of this book to get the URLs for downloading Node.js and NPM.

Electronic supplementary material The online version of this chapter (doi:10.1007/978-1-4842-2418-2_1) contains supplementary material, which is available to authorized users.

© Shankar Garg 2016
S. Garg, *Appium Recipes*, DOI 10.1007/978-1-4842-2418-2_1

Make sure you have not installed Node or Appium with `sudo` or you'll run into problems. Let's first check whether you have the latest versions of NPM and Node installed.

1. Type the following command in a terminal to check the Node version:

   ```
   node -v
   ```

2. Type the following command in a terminal to check the NPM version:

   ```
   npm -version
   ```

   Your terminal output should match Figure 1-1.

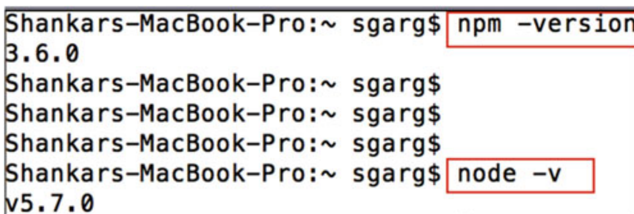


Figure 1-1. *Checking the versions of NPM and Node*

3. Type the following command in a terminal to install the Appium 1.5.0 release:

   ```
   npm install -g appium@1.5.0
   ```

   Note:

   1. appium@1.5.0 is to specifically download a particular version of Appium which is not the latest version.

   2. if you know the latest release of Appium is stable then you can use the command "npm install -g appium" to directly install latest version.

4. Observe the output in the terminal; it should look like Figures 1-2 through 1-4.

```
sgarg — node — 103×33
Shankars-MacBook-Pro:~ sgarg$ npm install -g appium@1.5.0
npm WARN deprecated mobile-json-wire-protocol@1.3.2: This package has been migrated into the 'appium-ba
se-driver' package
loadRequestedDeps → fetch ▌ ╢████████████░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░╟
```

Figure 1-2. *Appium download starting*

```
sgarg — bash — 123×45
Shankars-MacBook-Pro:~ sgarg$
Shankars-MacBook-Pro:~ sgarg$ npm install -g appium@1.5.0
npm WARN deprecated mobile-json-wire-protocol@1.3.2: This package has been migrated into the 'appium-base-driver' package
/usr/local/bin/authorize-ios -> /usr/local/lib/node_modules/appium/node_modules/.bin/authorize-ios
/usr/local/bin/appium -> /usr/local/lib/node_modules/appium/build/lib/main.js

> fsevents@1.0.12 install /usr/local/lib/node_modules/appium/node_modules/fsevents
> node-pre-gyp install --fallback-to-build

[fsevents] Success: "/usr/local/lib/node_modules/appium/node_modules/fsevents/lib/binding/Release/node-v47-darwin-x64/fse.n
ode" is installed via remote

> appium-chromedriver@2.8.0 install /usr/local/lib/node_modules/appium/node_modules/appium-chromedriver
> node install-npm.js

info Chromedriver Install Installing Chromedriver version '2.21' for platform 'mac' and architecture '32'
info Chromedriver Install Opening temp file to write chromedriver_mac32 to...
info Chromedriver Install Downloading http://chromedriver.storage.googleapis.com/2.21/chromedriver_mac32.zip...
info Chromedriver Install Writing binary content to /var/folders/n2/gp3xz20x5l12ncbcln433f_hm9yrgq/T/2016422-14297-eifn1u/c
hromedriver_mac32.zip...
info Chromedriver Install Extracting /var/folders/n2/gp3xz20x5l12ncbcln433f_hm9yrgq/T/2016422-14297-eifn1u/chromedriver_mac
32.zip to /var/folders/n2/gp3xz20x5l12ncbcln433f_hm9yrgq/T/2016422-14297-eifn1u/chromedriver_mac32
info Chromedriver Install Creating /usr/local/lib/node_modules/appium/node_modules/appium-chromedriver/chromedriver/mac...
info Chromedriver Install Copying unzipped binary, reading from /var/folders/n2/gp3xz20x5l12ncbcln433f_hm9yrgq/T/2016422-14
297-eifn1u/chromedriver_mac32/chromedriver...
info Chromedriver Install Writing to /usr/local/lib/node_modules/appium/node_modules/appium-chromedriver/chromedriver/mac/c
hromedriver...
info Chromedriver Install /usr/local/lib/node_modules/appium/node_modules/appium-chromedriver/chromedriver/mac/chromedriver
 successfully put in place

> appium-selendroid-driver@1.2.2 install /usr/local/lib/node_modules/appium/node_modules/appium-selendroid-driver
> node ./bin/install.js

info SelendroidInstaller Ensuring /usr/local/lib/node_modules/appium/node_modules/appium-selendroid-installer/selendroid/do
wnload exists
info SelendroidInstaller Downloading Selendroid standalone server version 0.17.0 from https://github.com/selendroid/selendr
oid/releases/download/0.17.0/selendroid-standalone-0.17.0-with-dependencies.jar --> /usr/local/lib/node_modules/appium/node
_modules/appium-selendroid-installer/selendroid/download/selendroid-server.jar
info SelendroidInstaller Writing binary content to /usr/local/lib/node_modules/appium/node_modules/appium-selendroid-instal
ler/selendroid/download/selendroid-server.jar
info SelendroidInstaller Selendroid standalone server downloaded
info SelendroidInstaller Determining AndroidManifest location
info SelendroidInstaller Determining server apk location
info SelendroidInstaller Extracting manifest and apk to /usr/local/lib/node_modules/appium/node_modules/appium-selendroid-i
nstaller/selendroid/download
```

Figure 1-3. *Appium downloading packages*

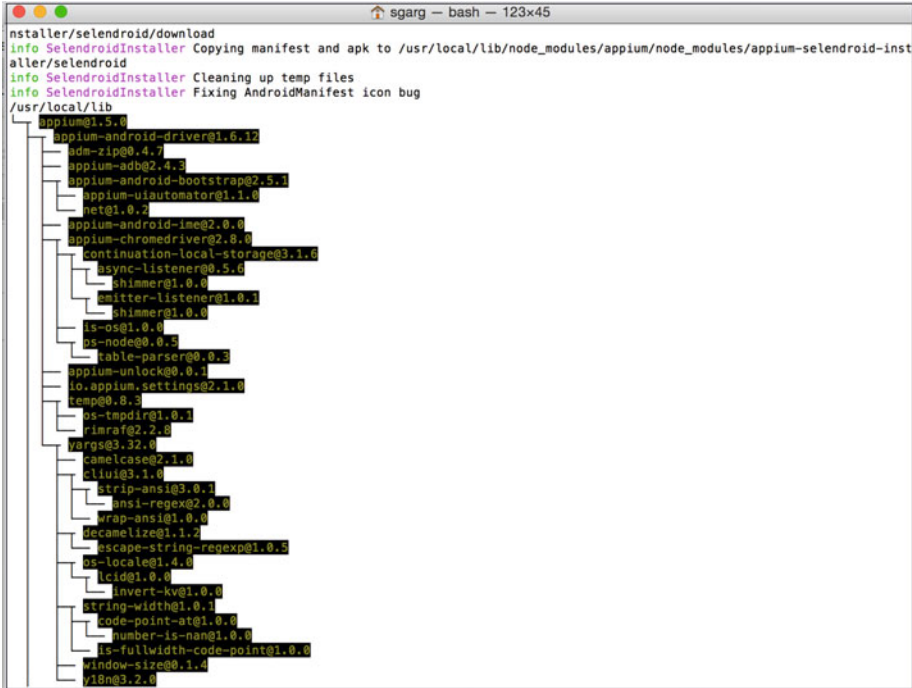


Figure 1-4. *Appium downloaded packages list*

5. Wait for NPM to download all the packages for Appium.

6. Type the following command in a terminal to check the Appium version:

```
appium -v
```

7. Type the following command in a terminal to start the Appium server, as shown in Figure 1-5:

```
appium
```

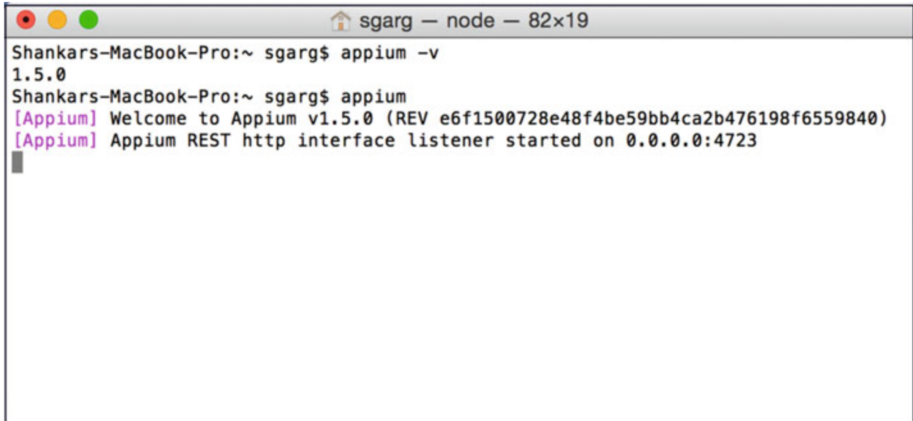


Figure 1-5. *Appium server running*

8. If you want to check whether all the dependencies for Appium are met, then type the following command in a terminal, which results in Figure 1-6:

```
appium-doctor
```

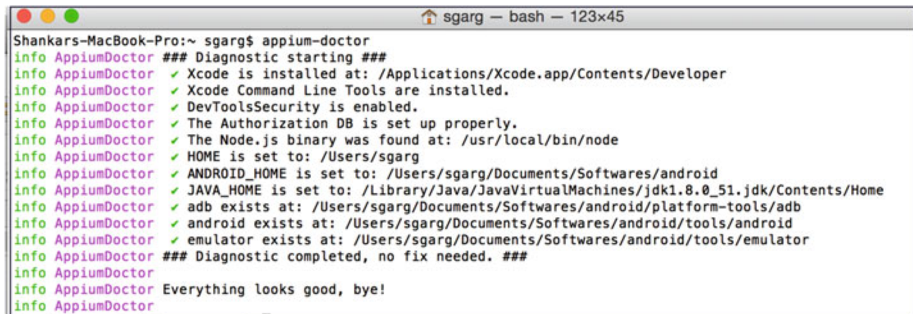


Figure 1-6. *AppiumDoctor*

How It Works

To start the Appium server, first you need to install Appium. NPM is the best package manager for installing Appium. Using the `-g` option while installing means Appium will be installed globally. The command to start the Appium server is `appium`. The Appium server is now ready for your use.

You can use AppiumDoctor to check that Appium installed with the correct configuration settings. Since Appium can be used for both platforms, the settings are platform-specific, such as `ANDROID_HOME` for Android and Xcode for iOS. To check the platform-specific dependencies, use `appium-doctor --ios` for iOS and `appium-doctor --android` for Android (Figure 1-7).

```
sgarg — bash — 93×30
Shankars-MacBook-Pro:~ sgarg$ appium-doctor --ios
info AppiumDoctor ### Diagnostic starting ###
info AppiumDoctor  ✔ Xcode is installed at: /Applications/Xcode.app/Contents/Developer
info AppiumDoctor  ✔ Xcode Command Line Tools are installed.
info AppiumDoctor  ✔ DevToolsSecurity is enabled.
info AppiumDoctor  ✔ The Authorization DB is set up properly.
info AppiumDoctor  ✔ The Node.js binary was found at: /usr/local/bin/node
info AppiumDoctor  ✔ HOME is set to: /Users/sgarg
info AppiumDoctor ### Diagnostic completed, no fix needed. ###
info AppiumDoctor
info AppiumDoctor Everything looks good, bye!
info AppiumDoctor
Shankars-MacBook-Pro:~ sgarg$
Shankars-MacBook-Pro:~ sgarg$
Shankars-MacBook-Pro:~ sgarg$ appium-doctor --android
info AppiumDoctor ### Diagnostic starting ###
info AppiumDoctor  ✔ ANDROID_HOME is set to: /Users/sgarg/Documents/Softwares/android
info AppiumDoctor  ✔ JAVA_HOME is set to: /Library/Java/JavaVirtualMachines/jdk1.8.0_51.jdk/C
ontents/Home
info AppiumDoctor  ✔ adb exists at: /Users/sgarg/Documents/Softwares/android/platform-tools/a
db
info AppiumDoctor  ✔ android exists at: /Users/sgarg/Documents/Softwares/android/tools/androi
d
info AppiumDoctor  ✔ emulator exists at: /Users/sgarg/Documents/Softwares/android/tools/emula
tor
info AppiumDoctor ### Diagnostic completed, no fix needed. ###
info AppiumDoctor
info AppiumDoctor Everything looks good, bye!
info AppiumDoctor
Shankars-MacBook-Pro:~ sgarg$
```

Figure 1-7. *AppiumDoctor's platform-specific output*

1-2. Run Appium via a GUI App

Problem

You are not comfortable running Appium via terminal. Since the Appium team also supports a GUI app that is available for both the Windows and Mac operating systems, you want to use the GUI app to run the Appium server.

Solution

You need the latest release of the Appium GUI app, which can be downloaded from `https://bitbucket.org/appium/appium.app/downloads/`. The latest release as of this writing is 1.5.3. Once the app is downloaded, just follow the prompts to install the app. It's a straightforward process.

1. Open the GUI app by clicking the app icon. The user interface (UI) shown in Figure 1-8 appears.

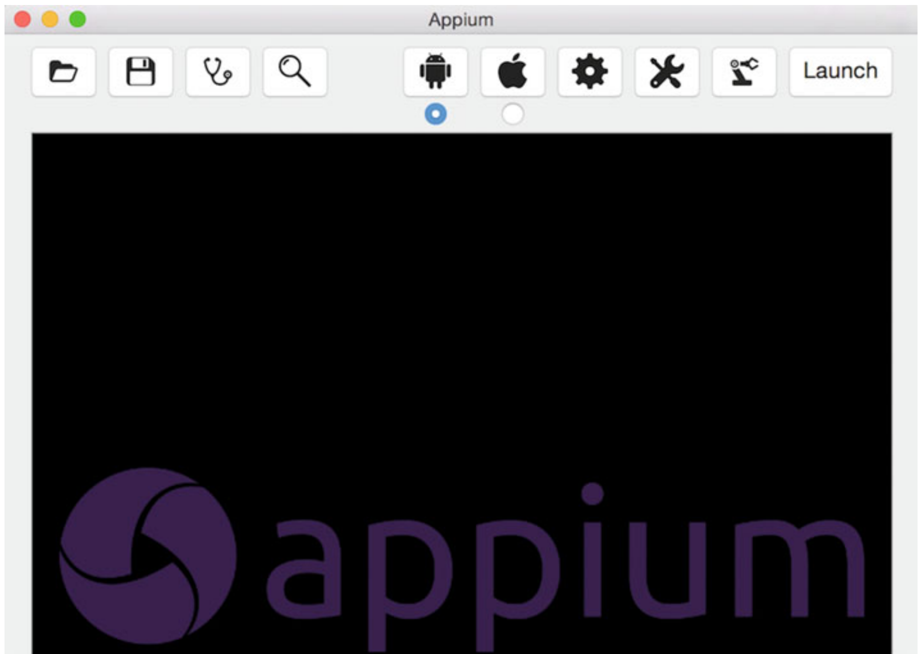


Figure 1-8. *Appium GUI app*

2. To start the Android server, click the Android icon in the top menu and click Launch. The Appium server for Android will start (Figure 1-9).

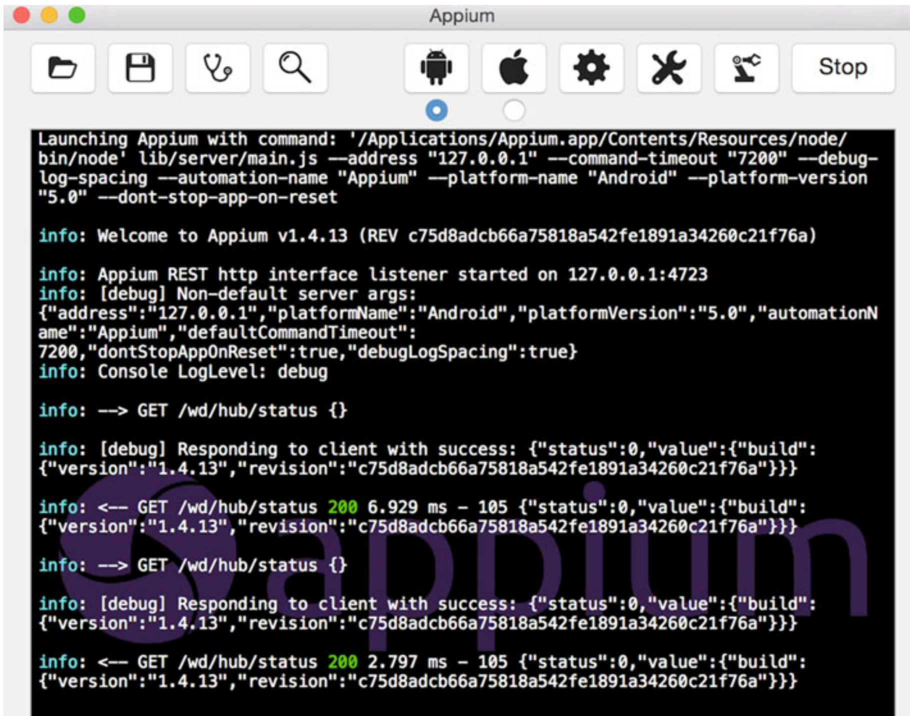


Figure 1-9. *Appium GUI app, Android*

3. To start the iOS server, click the iOS icon in the top menu and click Launch. The Appium server for iOS will start (Figure 1-10).

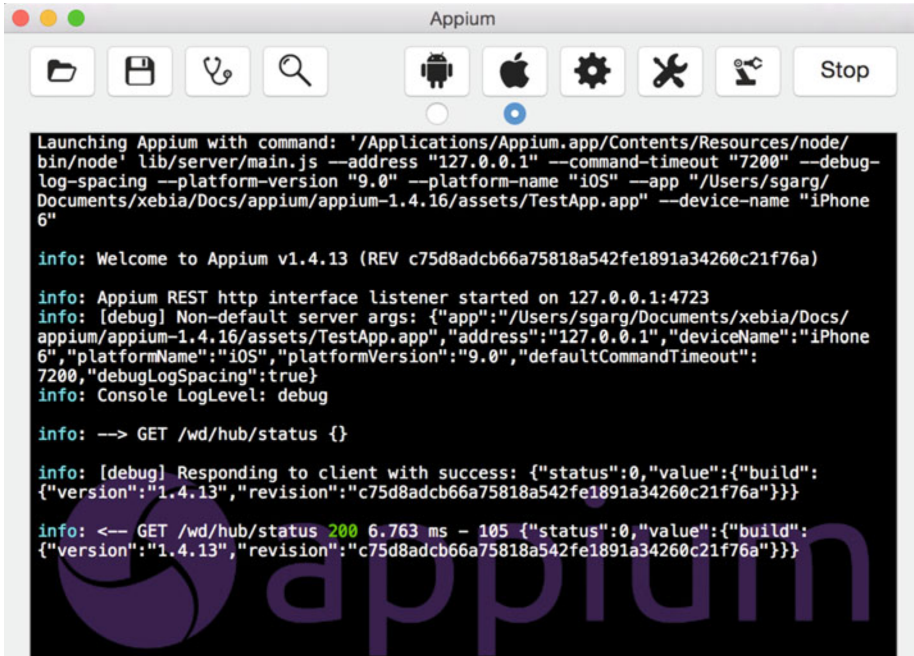


Figure 1-10. *Appium GUI app, iOS*

4. If you want to check whether all the dependencies for Appium are met, then click the stethoscope icon in the top-left corner, as shown in Figure 1-11.

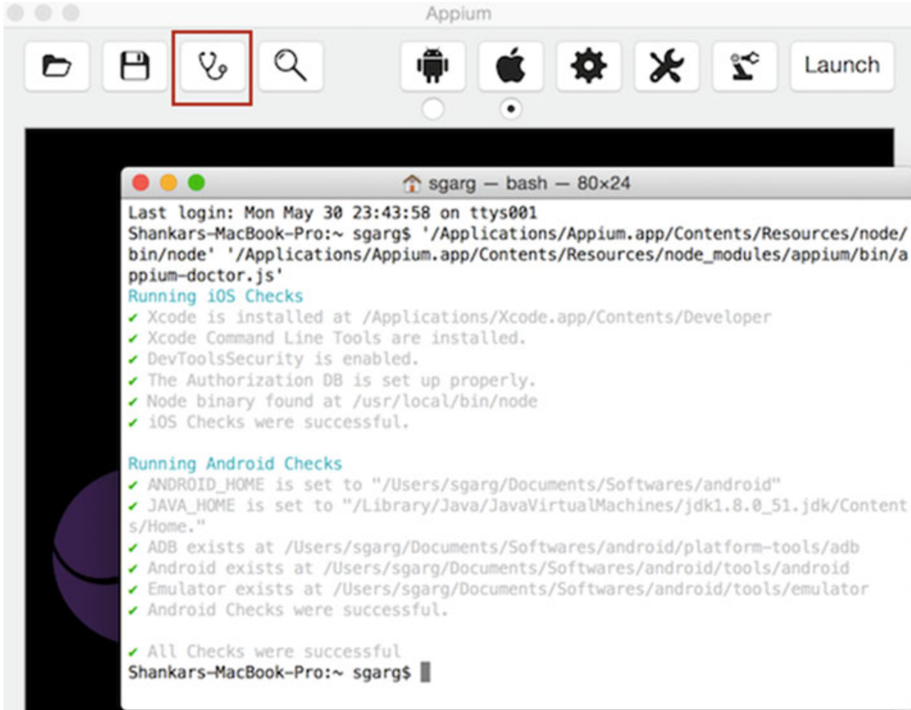


Figure 1-11. *Appium GUI app, checking whether all the dependencies for Appium are met*

How It Works

Once the Appium app is installed, you can run the Appium server. All you need to do is select which platform you want to run the Appium server for and then click Launch. The appropriate Appium server will be launched.

The Appium GUI app also supports AppiumDoctor, which helps you check whether all the dependencies for Appium are set. For this, just click the stethoscope icon in the top-left menu, and AppiumDoctor will run all the checks and let you know the status in a terminal window.

1-3. Understand Capabilities in Appium

Problem

Appium is based on Selenium; in a way, it's an extension of Selenium. Most of the commands that you use in Selenium work with Appium also (provided those Selenium commands make sense for mobile automation), so let's talk about how Appium extends Selenium.

Appium works in a client-server architecture. The client (test case) requests features that a session should support. The client and server use JavaScript Object Notation (JSON) objects with predefined properties when describing the features that a test case is asking a session to support. These JSON objects and their properties are called *desired capabilities*. (For more information, please go to http://appium.io/slate/en/master/?java#about-appium)

You want to see how to set the desired capabilities for mobile automation.

Solution

You can set the desired capabilities at the server level or at the client level. Capabilities at the server level can be set using the command line or the Appium GUI app, and at client level they will be set in the test case via code.

Capabilities via a GUI App

To use the GUI app for iOS, click the iOS icon and choose the capabilities you want, as shown in Figure 1-12.

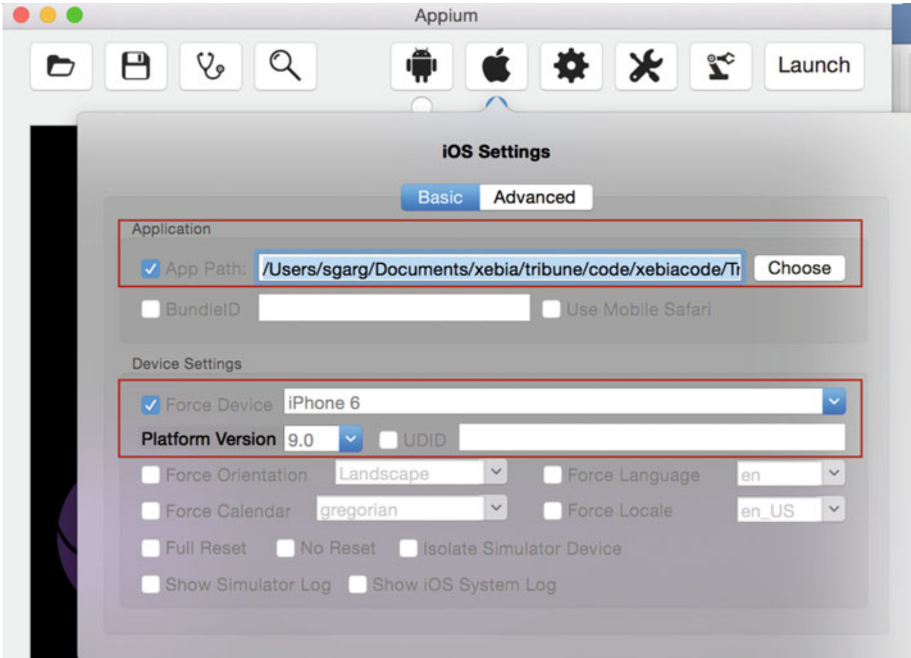


Figure 1-12. *Appium iOS capabilities*

To use the GUI app for Android, click the Android icon and choose the capabilities you want, in Figure 1-13.

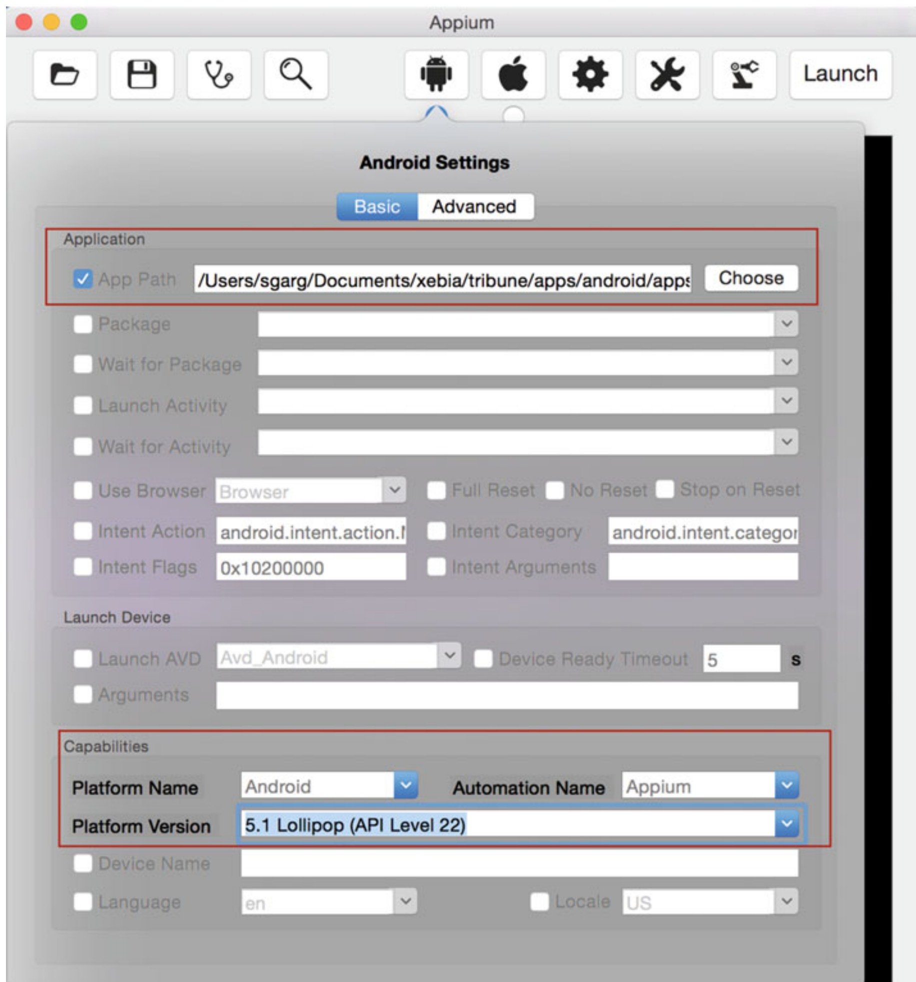


Figure 1-13. *Appium Android capabilities*

Capabilities via a Terminal

To choose the capabilities via a terminal, follow these steps:

1. Open a terminal and type the following command to check all the capabilities available via a terminal (Figure 1-14):

    ```
    appium -help
    ```

```
sgarg — bash — 97×26
Last login: Sun Jun  5 22:44:24 on ttys002
Shankars-MacBook-Pro:~ sgarg$ appium -help
usage: appium [-h] [-v] [--shell] [--ipa IPA] [-a ADDRESS] [-p PORT]
              [-ca CALLBACKADDRESS] [-cp CALLBACKPORT] [-bp BOOTSTRAPPORT]
              [-r BACKENDRETRIES] [--session-override] [-l] [-g LOG]
              [--log-level {info,info:debug,info:info,info:warn,info:error,warn,warn:debug,warn:i
nfo,warn:warn,warn:error,error,error:debug,error:info,error:warn,error:error,debug,debug:debug,de
bug:info,debug:warn,debug:error}]
              [--log-timestamp] [--local-timezone] [--log-no-colors]
              [-G WEBHOOK] [--safari] [--default-device] [--force-iphone]
              [--force-ipad] [--tracetemplate AUTOMATIONTRACETEMPLATEPATH]
              [--instruments INSTRUMENTSPATH] [--nodeconfig NODECONFIG]
              [-ra ROBOTADDRESS] [-rp ROBOTPORT]
              [--selendroid-port SELENDROIDPORT]
              [--chromedriver-port CHROMEDRIVERPORT]
              [--chromedriver-executable CHROMEDRIVEREXECUTABLE]
              [--show-config] [--no-perms-check]
              [--command-timeout DEFAULTCOMMANDTIMEOUT] [--strict-caps]
              [--isolate-sim-device] [--tmp TMPDIR] [--trace-dir TRACEDIR]
              [--debug-log-spacing] [--suppress-adb-kill-server]
              [--async-trace] [--webkit-debug-proxy-port WEBKITDEBUGPROXYPORT]
              [--default-capabilities DEFAULTCAPABILITIES] [-k]
              [--platform-name PLATFORMNAME]
              [--platform-version PLATFORMVERSION]
              [--automation-name AUTOMATIONNAME] [--device-name DEVICENAME]
              [--browser-name BROWSERNAME] [--app APP] [-lt LAUNCHTIMEOUT]
```

Figure 1-14. *Appium help via terminal*

2. Once you know which capabilities you need to set, type the following command to run the Android server (Figure 1-15):

```
appium --platform-name 'iOS'  --platform-version
'9.0'  --browser-name  '' --device-name 'ANDROID'
```

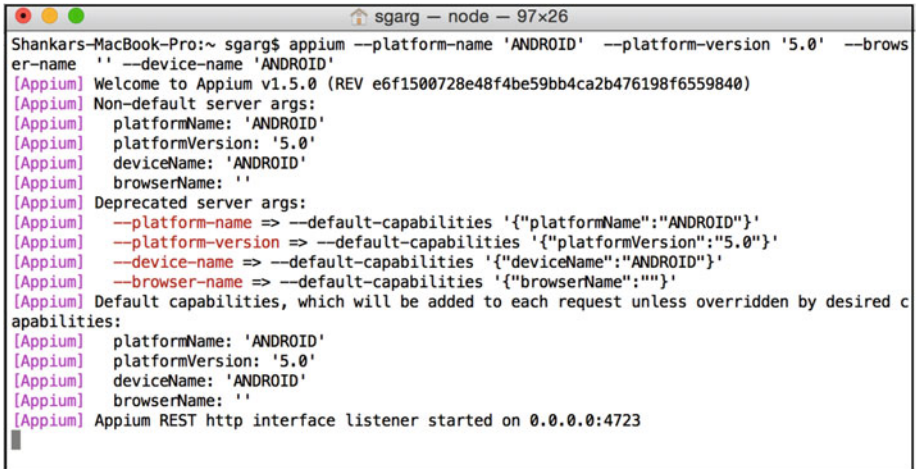


Figure 1-15. *Appium Android server via a terminal*

3. Type the following command to run the iOS server via a terminal (Figure 1-16):

```
appium --platform-name 'iOS'  --platform-version '9.0'
--browser-name  '' --device-name 'iPhone 6'
```

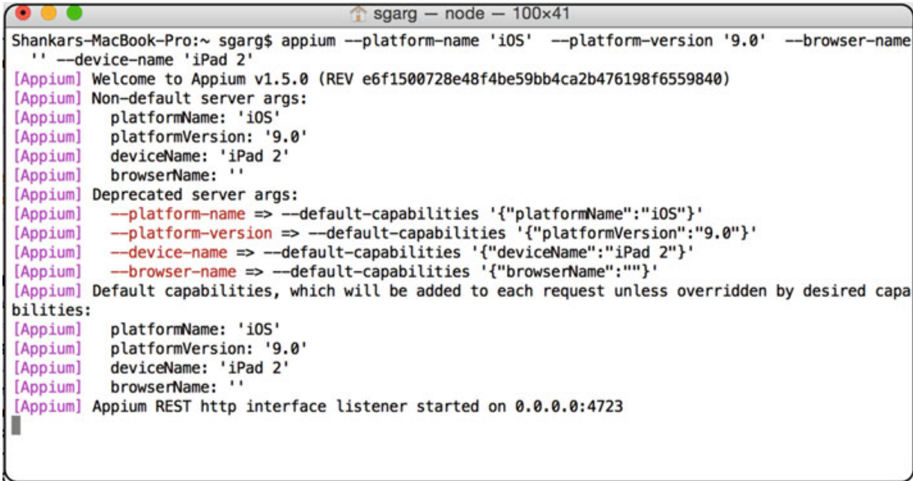


Figure 1-16. *Appium iOS server via a command line*

Capabilities via Code

Review the blog at https://shankargarg.wordpress.com/2016/02/25/create-an-appium-project-by-integrating-appium-eclipse-maven-testng/ to see how to create a sample Appium project.

Then follow these steps:

1. Use the following code when initializing the Appium driver object for the iOS capabilities:

```
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("browserName", "");
caps.setCapability("platformVersion", "9.0");
caps.setCapability("platformName", "iOS");
caps.setCapability("platform", "MAC");
caps.setCapability("deviceName", "iPhone 6");

// relative path to app/ipa file
final File classpathRoot = new File(System.
getProperty("user.dir"));
final File appDir = new File(classpathRoot, "src/test/
resources/apps/");
```

```
final File app = new File(appDir, " TestApp.app");
        caps.setCapability("app", app.
getAbsolutePath());

// initializing driver object
driver = new IOSDriver(new URL("http://127.0.0.1:4723/
wd/hub"), caps);
```

2. Use the following code when initializing the Appium driver object for the Android capabilities:

```
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("platform", "ANDROID");
caps.setCapability("platformVersion", "5.0");
caps.setCapability("deviceName", "ANDROID");
caps.setCapability("browserName", "");

// relative path to apk file
final File classpathRoot = new File(System.
getProperty("user.dir"));
final File appDir = new File(classpathRoot, "src/test/
resources/apps/");
final File app = new File(appDir, "ApiDemos-debug.apk");
        caps.setCapability("app", app.getAbsolutePath());

// initializing driver object
driver = new AndroidDriver(new
URL("http://127.0.0.1:4723/wd/hub"), caps);
```

How It Works

Desired capabilities are a set of keys and values (that is, a map or hash) sent to the Appium server to tell the server what kind of automation session you're interested in.

When initiating a Selenium driver, you need to mention the browser that you need to invoke. Similarly, for Appium you need to mention the platform, such as Android or iOS, and platform version, such as iOS 9.3 or Android 5.0.

Desired capabilities can be set at two levels:

- *Server level (GUI app or terminal)*: Capabilities mentioned while starting the Appium server will be added to each request unless they are overridden by the desired capabilities sent by the test case. For example, if you specify `iPhone 6` in the Appium server and start a generic Appium client without any device, then the test case will automatically run on iPhone 6.
- *Client level (test case)*: Capabilities mentioned while initiating the Appium client will override the capabilities of the Appium server. For example, if the Appium server has `iPad 2` as the device but you are sending `iPhone 6` in the test case request, then the test cases will run on iPhone 6.

www.allitebooks.com

Table 1-1 lists some of the desired capabilities that you will use most often.

Table 1-1. *Common Desired Capabilities*

Capability	Description	Values
platformName	Which mobile OS platform to use.	iOS, Android, or FirefoxOS
platformVersion	Mobile OS version.	Examples: 9.0, 5.0
deviceName	The kind of mobile device or emulator to use.	Examples: ANDROID, iPhone 6
app	The absolute local path or remote HTTP URL to an .ipa or .apk file, or a .zip containing one of these.	Example: /abs/path/to/my.apk
browserName	The name of mobile web browser to automate. This should be an empty string if automating an app instead.	Safari for iOS Chrome, Chromium, or Browser for Android
platformName	OS platform.	iOS, Android
platformVersion	OS version.	9.0, 9.1, 8.4, and so on, for iOS 5.0, 6.0, 4.4, and so on, for Android
deviceName	Mobile device ID.	iPhone 6, iPad 2, and so on, for iOS ANDROID, and so on, for Android

For an exhaustive list of all capabilities, please refer to https://github.com/appium/appium/blob/master/docs/en/writing-running-appium/caps.md.

CHAPTER 2

Finding Mobile Elements

In this chapter, you will learn how to do the following:

- Traverse with Appium Inspector
- Explore UI Automator Viewer
- Find elements by their accessibility ID
- Find elements using `IOSUIAutomation`
- Find elements using `AndroidUIAutomator`
- Inspect iOS mobile web elements
- Inspect Android mobile web elements

In the previous chapter, you learned how to set up and run Appium, but for mobile automation, that's not sufficient. You also need to know how to find mobile elements so you can interact with those elements to perform desired actions.

Since Appium is an extension of Selenium, most of the principles of finding elements in Selenium apply to finding elements in Appium. The only thing that changes is the context: i.e. mobile. So, in this chapter, you'll understand how to find mobile elements.

Before going further, make sure to download the project from the book's GitHub repository: `https://github.com/ShankarGarg/AppiumBook/tree/master/AppiumRecipesBook`.

2-1. Traverse with Appium Inspector

Problem

You want to inspect the user interface (UI) of an application to find the layout hierarchy and view the properties associated with the elements.

© Shankar Garg 2016
S. Garg, *Appium Recipes*, DOI 10.1007/978-1-4842-2418-2_2

Solution

With the Appium graphical user interface (GUI) app, you can use a built-in utility Appium Inspector to find elements for native iOS apps.

1. In Appium's iOS Settings, provide the path of the iOS app that you want to find elements for (Figure 2-1). A sample iOS app is saved in the `src/test/resources/apps/` folder of the code that you have checked out for this chapter (`AppiumRecipesBook`).

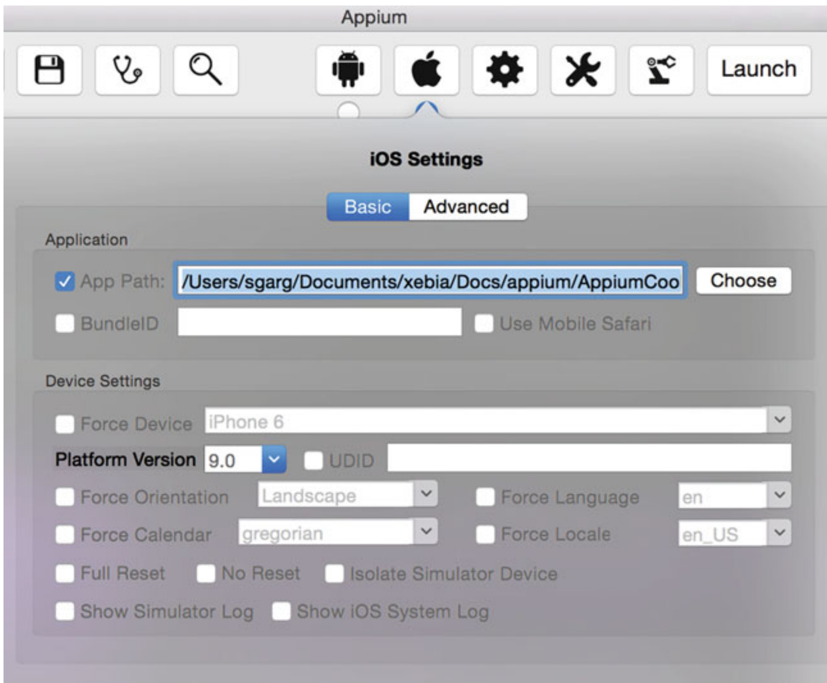


Figure 2-1. *App path of the .app file for iOS*

2. In Appium's General Settings, select the Prelaunch Application check box, as shown in Figure 2-2.

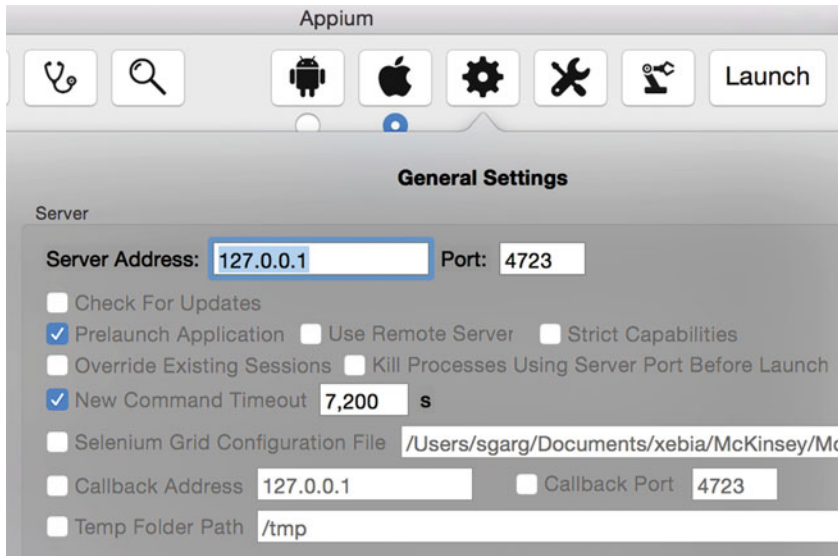


Figure 2-2. *Selecting the Prelaunch Application check box*

■ **Note** If Prelaunch Application is not selected, Appium will launch the app when you click the Appium Inspector icon.

---

3. Click the Launch button to launch the Appium server. Wait for the Appium server to start and wait for Appium to launch the iOS simulator with the desired app opened.

4. Click the magnifying glass icon in the top-left corner in the Appium GUI app (Figure 2-3). The Appium Inspector window will open with the application's current state captured.

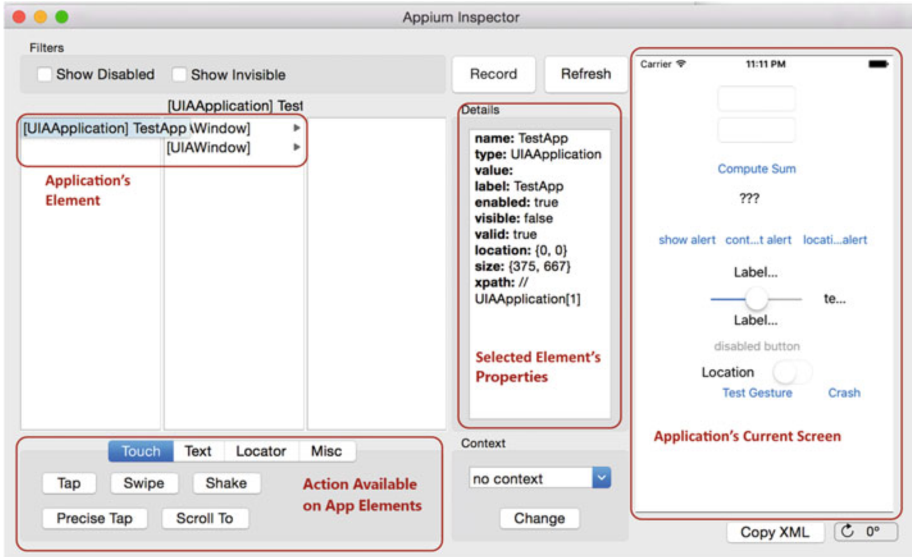


Figure 2-3. *Appium Inspector*

5. Now you want to find the properties of the first text box available onscreen. Double-click the element on the screen in the right panel in the Appium Inspector window.

6. Once you select the element in the right panel, all the properties of that element will be displayed in the middle panel, and the hierarchy will be displayed in the left panel, as shown in Figure 2-4.

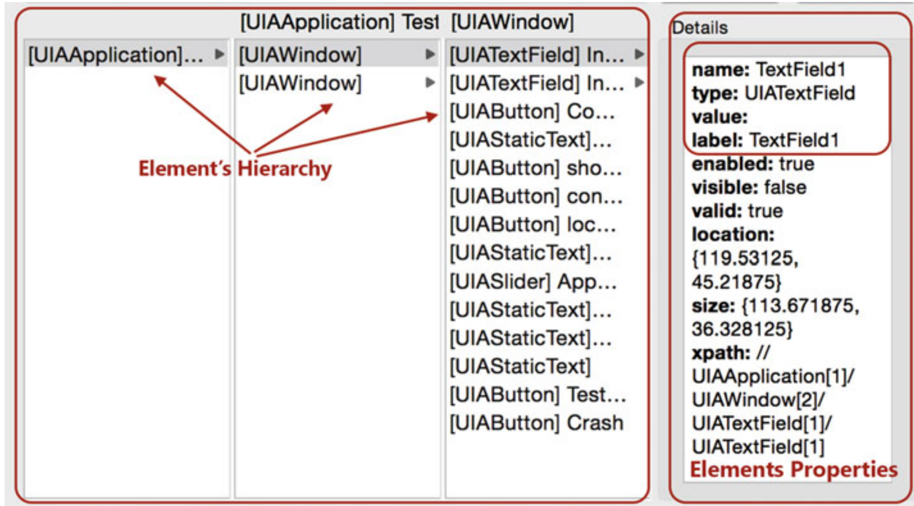


Figure 2-4. *Element properties in iOS*

7. For this particular text field, you can check the attributes such as `name`, `type`, `label`, `xpath`, and so on, and you can use these attributes in test scripts.

8. You can select elements in the hierarchy viewer, and they will be selected in the right panel.

9. Now if you select the third element from the bottom in the `[UIAwindow]` area, then details of that element will be visible in the middle panel, and that element will also be selected in the right panel, as shown in Figure 2-5.

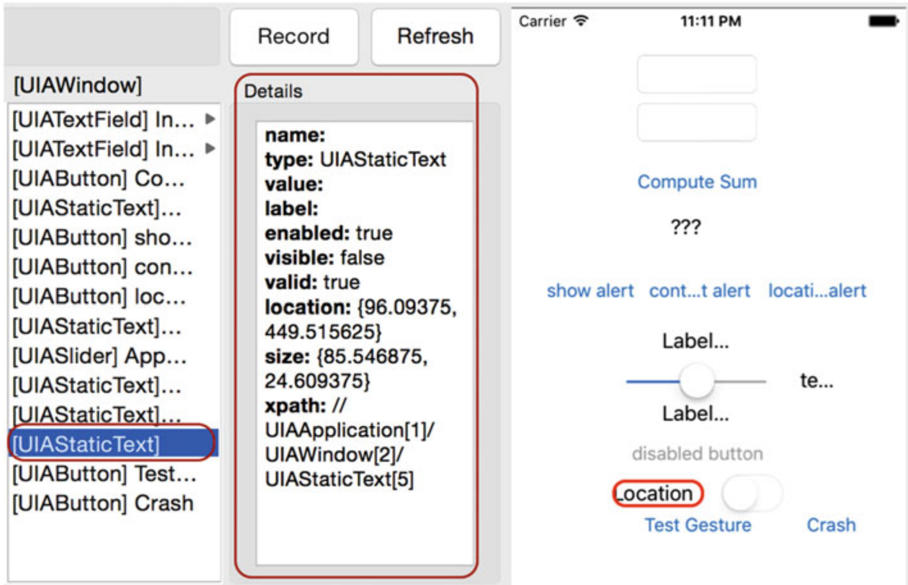


Figure 2-5. *Appium Inspector details*

How It Works

You can access Appium Inspector by clicking the magnifying glass icon next to the Launch button. The Appium server must be running with an app open or Appium Inspector will not work. Once Appium Inspector is up, then you can select an element to check its various attributes such as `name`, `value`, `xpath`, and so on.

Appium Inspector is used to accomplish the following:

- Identify and understand the element hierarchy
- Find attributes of the element/object
- Record your manual actions with the app

■ **Note** Appium Inspector is best suited for iOS native apps. For Android native apps, you will use UI Automator Viewer, which I will cover in the next recipe.

2-2. Explore UI Automator Viewer

Problem

Although Appium has a built-in utility Appium Inspector for identifying elements, it does not work properly and efficiently for Android native apps. You want to use UIAutomatorViewer to find elements in an Android native app.

Solution

To use UI Automator Viewer, the Android software development kit (SDK) must be installed, and the path must be updated for the Android SDK.

■ **Note** For more information on this topic, please follow these instructions:

`https://shankargarg.wordpress.com/2016/02/25/setup-android-sdk-and-android-emulators/`

`https://shankargarg.wordpress.com/2016/02/25/setup-genymotion-android-emulators-on-mac-os/`

For Android native apps, you can use UI Automator Viewer by following these steps:

1. Open the Genymotion emulator and install the `ApiDemos-debug.apk` app on it.
2. Go to the location where you downloaded the Android SDK, go to the Tools folder, and double-click `uiautomatorviewer`.

   Or, if the Android SDK path is set, go to a terminal, type `uiautomatorviewer`, and press Enter.

Your screen should match Figure 2-6.

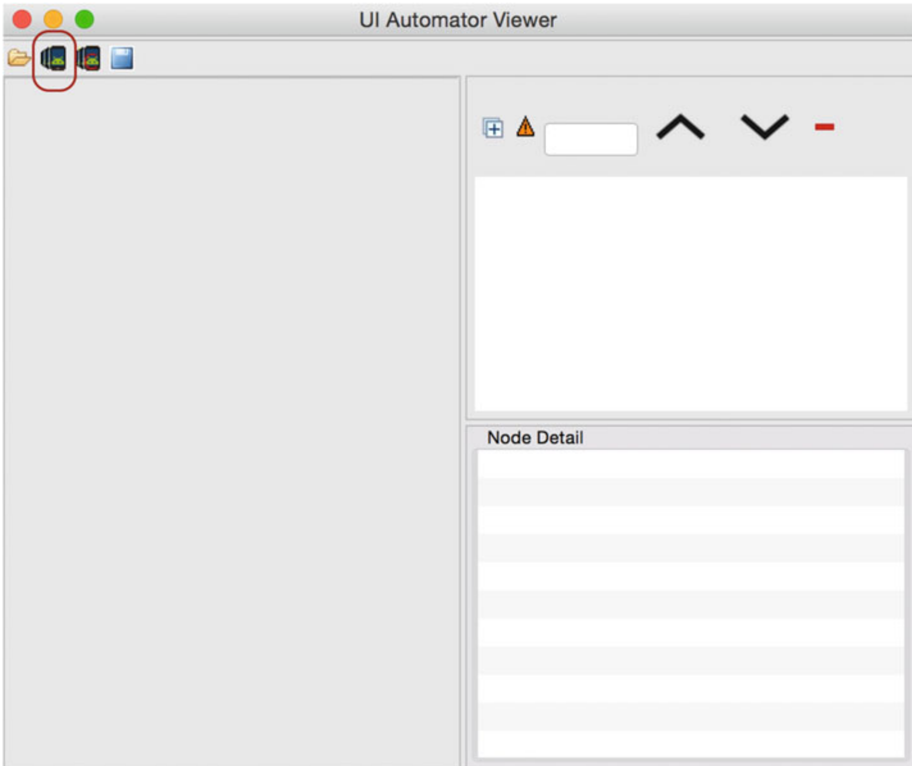


Figure 2-6. *UI Automator Viewer*

3. Clicking the devices icon on the left takes a snapshot of the screen that's open on the device/emulator, as shown in Figure 2-7.

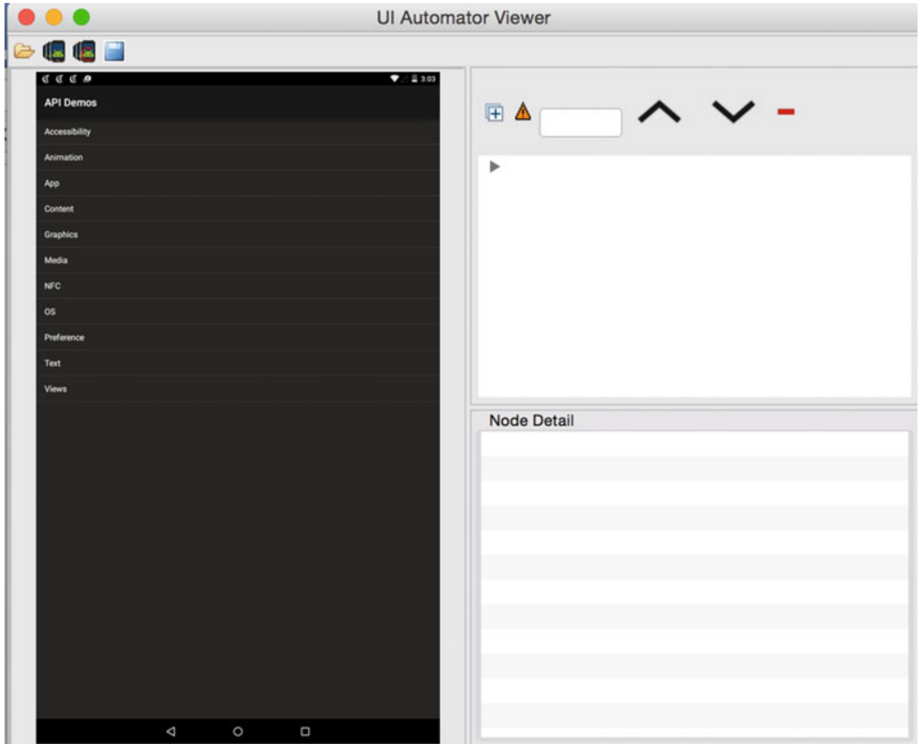


Figure 2-7. *UI Automator Viewer default screen*

4. Now you want to find properties of the Accessibility button (the first option available on the screen). Double-click the element on the screen in the left panel in the UI Automator Viewer window.

5. Once you select the element in the left panel, all the properties of that element will be displayed in the bottom-right panel, and the hierarchy will be displayed in top-right panel, as shown in Figure 2-8.

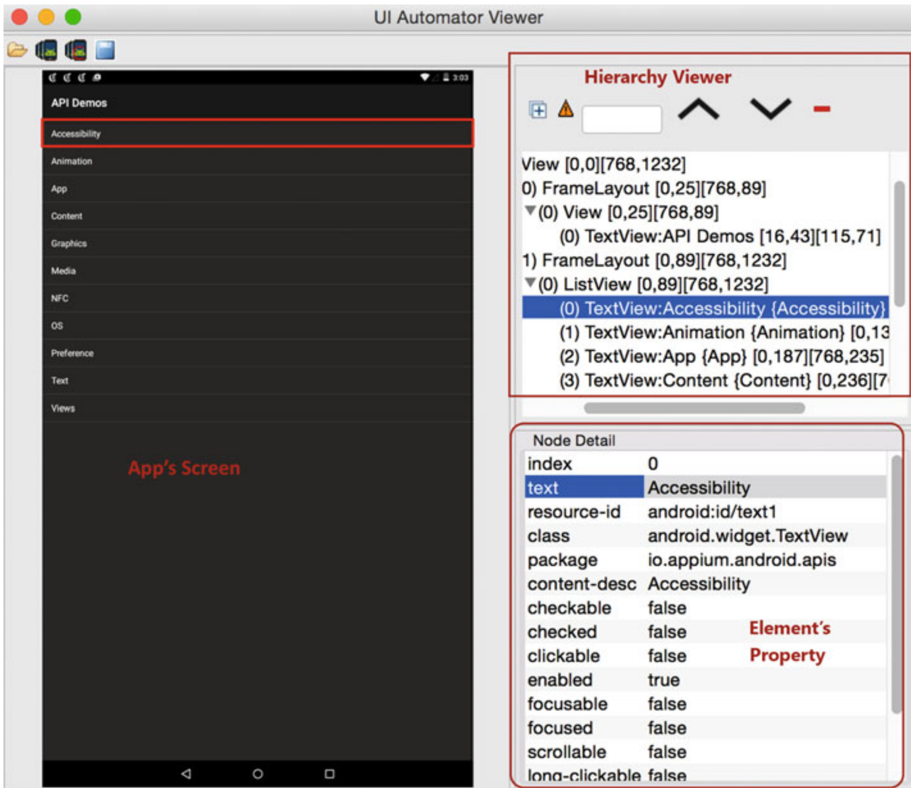


Figure 2-8. *UI Automator Viewer details screen*

6. For this particular field, you can check the attributes such as `text`, `resource-id`, `class`, `content-desc`, and so on, and you can use these attributes in test cases.

7. You can select elements in the hierarchy viewer also. They will be selected in the left panel, and their properties will be displayed in the bottom-right panel.

8. Now if you select the second `TextView` in the top-right window (Figure 2-9), you will see details of that element in the bottom-right panel; that element will also be selected in the left panel.

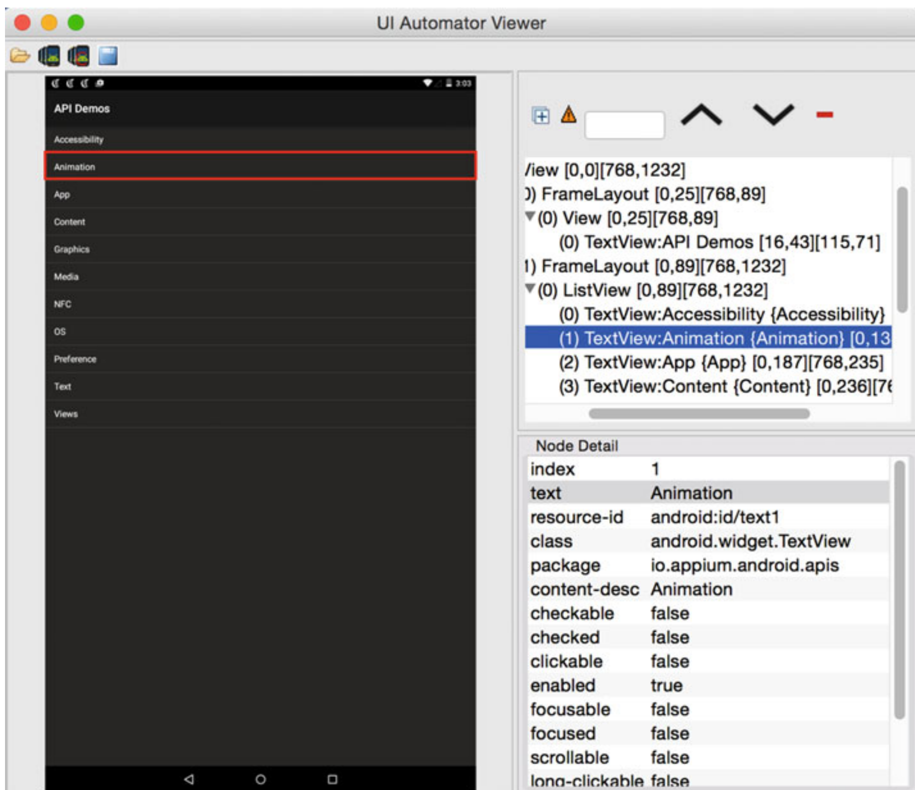


Figure 2-9. UI Automator Viewer details screen

How It Works

UI Automator Viewer is an inspector tool provided by Google that lets you inspect the UI of an application to find the layout hierarchy and view the properties associated with the controls. It will work only if an emulator/device is live and an app is opened in the emulator. Once UI Automator Viewer is up, then a particular element can be selected to check its various attributes such as `resource-id`, `class`, and so on.

2-3. Find Elements by Their Accessibility ID

Problem

To interact with elements to perform actions, you need to first find the elements. Since Appium extends Selenium, all generic locator strategies such as `name`, `id`, `xpath`, and so on, are available in Appium, and these can be used effectively in Appium. In this book, you will focus on locator strategies specific to Appium.

Accessibility ID is one strategy that is available for both the Android and iOS platforms and is very stable. Let's understand to use accessibility ID to find elements.

Solution

Android

As explained in the previous recipe, you can use UI Automator Viewer for the API Demo Android application.

1. Select any element in the left panel and observe the `text` and `content-desc` properties in the bottom-right panel, as shown in Figure 2-10.

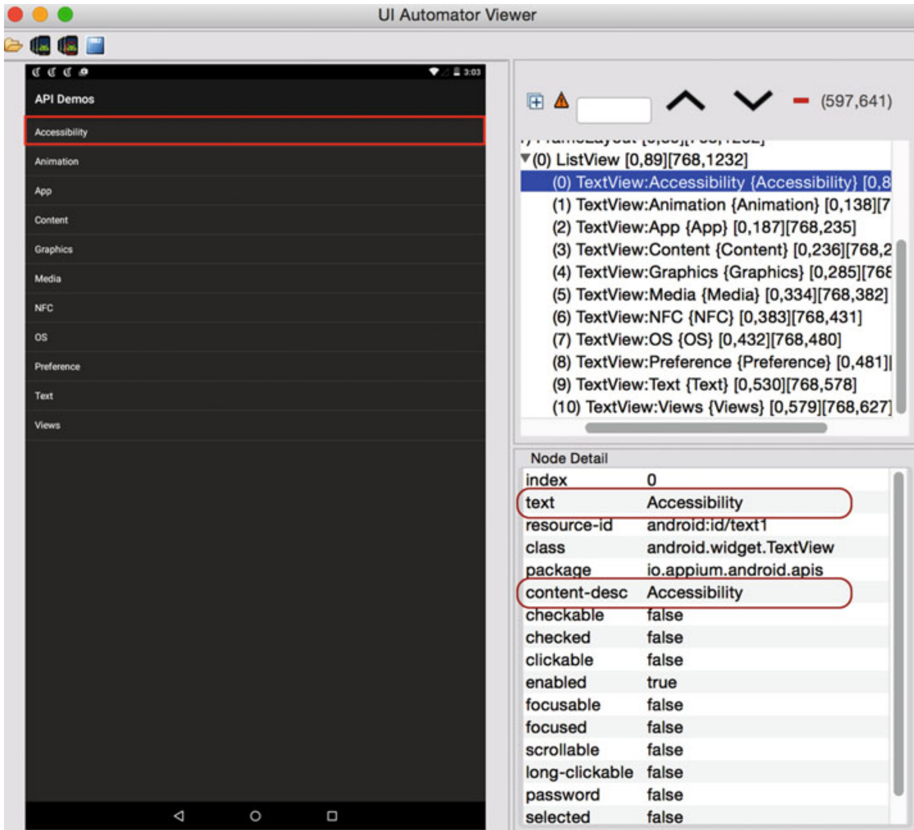


Figure 2-10. *Accessibility ID for Android*

2. In the `AppiumRecipesBook` project, go to the `AppiumSampleTestCaseAndroid` class and use the following code to interact with the first menu option:

```
// click on Accessibility link
wait.until(ExpectedConditions.presenceOfElementLocated(
MobileBy.AccessibilityId("Accessibility")));
driver.findElement(MobileBy.AccessibilityId("Accessibil
ity")).click();

// click on 'Accessibility Node Querying' link
wait.until(ExpectedConditions.presenceOfElementLoc
ated(MobileBy.AccessibilityId("Accessibility Node
Querying")));
driver.findElement(MobileBy.AccessibilityId("Accessibil
ity Node Querying")).click();
```

iOS

As explained in the previous recipe, let's use Appium Inspector for the TestApp iOS application.

1. Select the first text box in the right panel and observe the name properties in the middle panel, as shown in Figure 2-11.

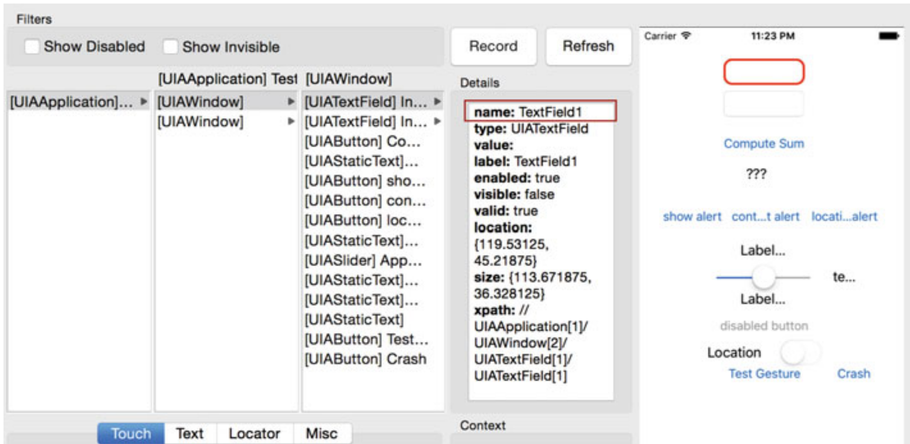


Figure 2-11. Accessibility ID for iOS

2. In the AppiumRecipesBook project, go to the AppiumSampleTestCaseiOS class and use the following code to interact with the two text boxes present in the app:

```
//enter data in first text box
wait.until(ExpectedConditions.presenceOfElementLocated
(MobileBy.AccessibilityId("TextField1")));
driver.findElement(MobileBy.AccessibilityId("TextFie
ld1")).sendKeys("AppiumBook");
```

```
//enter data in second text box
wait.until(ExpectedConditions.presenceOfElementLocated
(MobileBy.AccessibilityId("TextField2")));
driver.findElement(MobileBy.AccessibilityId("TextFie
ld2")).sendKeys("First TC");
```

How It Works

Accessibility identifiers are identifiers that app developers attach to important elements so that people with disabilities can meaningfully interpret the UI. So, you can expect that most of the elements that are important to end users will have an accessibility identifier defined, thus making it one of the best candidates of locator strategies.

The accessibility ID is generally the `name` or `content-desc` attribute of an element. Since `name` and `text` remain the same for both the Android and iOS platforms, the same accessibility ID can be used for both platforms, and therefore you can use one test case for both platforms.

At the same time, you need to be cautious because the `text`/`name` field can change a lot during the app life cycle, which will break both the Android and iOS test cases. However, ideally one fix should fix both test cases.

■ **Note** Some developers have used the name locator strategy extensively in their Appium tests, but it's deprecated now and soon will be deleted. (See `https://discuss.appium.io/t/why-is-name-locator-strategy-being-depreciated/7106`). Thus, it's advisable that you replace the name strategy with the accessibility ID.

2-4. Find Elements Using iOSUIAutomation

Problem

Using common strategies for both the Android and iOS platforms has its own advantages, but accessibility IDs are limited to elements that a user really interacts with such as buttons. What about elements that do not have any specific ID associated with them such as search results or catalog options?

Using XPaths for such elements would be very slow for native apps. You want to use an iOS-specific strategy called `iOSUIAutomation`, which is fast and reliable.

Solution

As explained in the previous recipe, let's use Appium Inspector for the TestApp iOS application.

1. Select the first text box in the right panel and observe the properties in the middle panel, as shown in Figure 2-12.

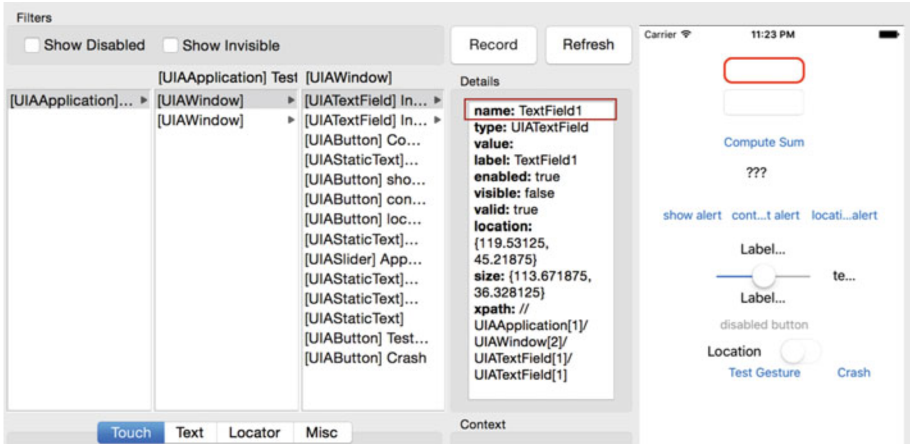


Figure 2-12. *iOSUIAutomation for iOS*

2. Go to the `AppiumSampleTestCaseiOS` class and use the following code to interact with the first and second text boxes present in the app:

```
// enter data in first text box
wait.until(ExpectedConditions.presenceOfElementLocated
(MobileBy.IosUIAutomation(".textFields()[0]")));
driver.findElement(MobileBy.IosUIAutomation(".
textFields()[0]")).sendKeys("1");

// enter data in second text box
wait.until(ExpectedConditions.presenceOfElementLocated
(MobileBy.IosUIAutomation(".textFields()[1]")));
driver.findElement(MobileBy.IosUIAutomation(".
textFields()[1]")).sendKeys("2");
```

3. Use the following code to interact with the Compute Sum button and then with the “???” label:

```
// click on compute Sum Button
driver.findElement(MobileBy.IosUIAutomation(".
buttons().firstWithPredicate(\"name=='ComputeSumButt
on'\")")).click();
// print value of ‘???’ label
System.out.println(driver.findElement(MobileBy.IosUIAut
omation(".staticTexts().firstWithPredicate(\"name=='Ans
wer'\")")).getText());
```

How It Works

`iOSUIAutomation` is an element-finding strategy powered by Apple specifically for the iOS platform. Since it is native to iOS, it's much faster than XPath, and it's much more powerful and flexible because it knows more platform-specific elements as compared to a generic XPath one.

`iOSUIAutomation` has predicates that allow you to select a specific element based on whether a condition is true.

If you are comfortable with XPath expressions or if you just copy the XPath expressions given by Appium Inspector, it's easy to convert XPath expressions to `iOSUIAutomation`. The rule of thumb for such a conversion is that the `UIAElementArray` numbering begins at 0, unlike XPath expressions where the index counting starts at 1. Take a look at these examples of simple expressions:

XPath: `/UIATableView[2]/UIATableCell[@label = 'Olivia'][1]`
iOS predicate: `tableViews()[1].cells().firstWithPredicate("label == 'Olivia' ")`

■ **Note** You can read more about iOS predicates at `http://appium.io/slate/en/master/?java#ios-predicate`.

2-5. Find Elements Using AndroidUIAutomator

Problem

You learned how to use the `iOSUIAutomation` locator strategy for iOS. Similarly, Let's learn to use `AndroidUIAutomator` for Android native apps.

Solution

As explained in previous recipes, you can use UI Automator Viewer for the API Demo Android application.

1. Select any element in the left panel and observe the properties in the bottom-right panel, as shown in Figure 2-13.

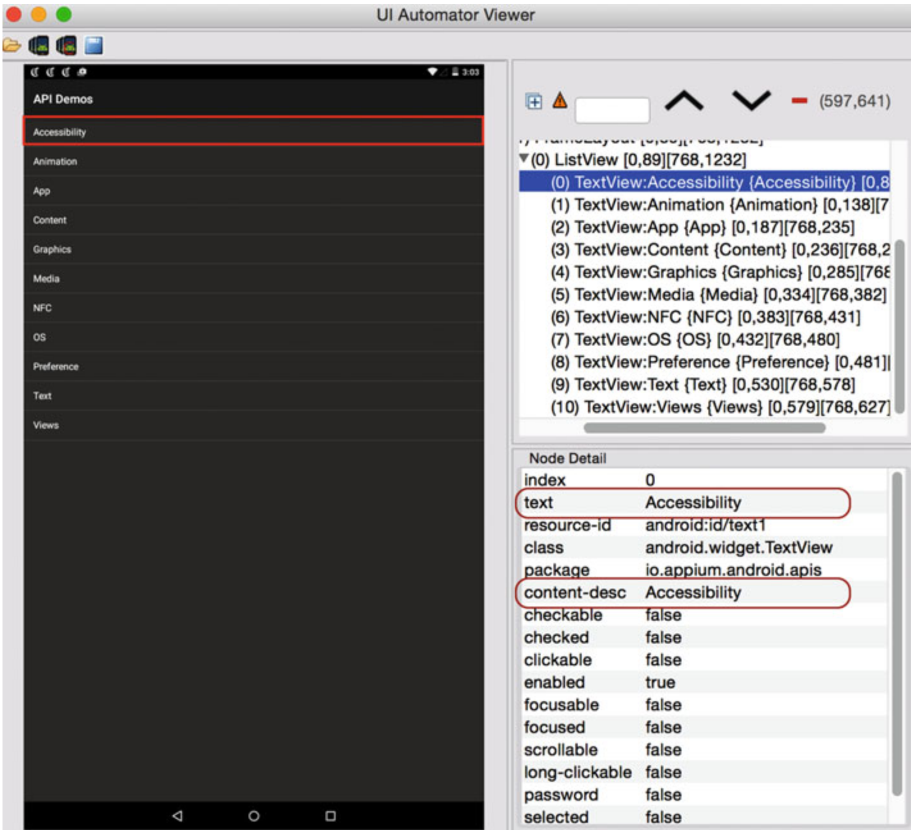


Figure 2-13. *Android UI Automator for Android*

2. As described in Chapter 1, go to the AppiumSampleTestCaseAndroid class and use the following code to interact with the first menu option:

```
//using classname and index
driver.findElement(MobileBy.AndroidUIAutomator("classNa
me(\"android.widget.TextView\").index(2)")).click();

//using text filter
driver.findElement(MobileBy.AndroidUIAutomator("text(\"
Alarm\")")).click();

driver.navigate().back();
driver.navigate().back();

//using content-desc
driver.findElement(MobileBy.AndroidUIAutomator
("description(\"Accessibility\")")).click();
```

How It Works

UISelector specifies the elements in the layout hierarchy for native apps, filtered by properties such as text value, content description, class name, and state information. You can also target an element by its location in a layout hierarchy using index(), but this should be considered as a last resort. If there is more than one matching widget, the first widget in the tree is selected.

■ **Note** You can read more about Android UI Automator and UISelector here: https://developer.android.com/reference/android/support/test/uiautomator/UiSelector.html

and here:

https://github.com/appium/appium/blob/master/docs/en/writing-running-appium/uiautomator_uiselector.md

---

www.allitebooks.com

2-6. Inspect iOS Mobile Web Elements

Problem

You want to find element properties of native elements for mobile web sites.

Solution

The following steps show how you can use the Safari developer plug-in to find iOS mobile web elements:

1. In your iOS simulator, go to Settings ➤ Safari ➤ Advanced and turn on Web Inspector (Figure 2-14).

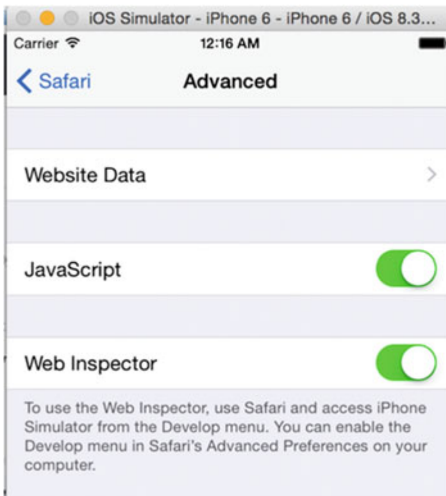


Figure 2-14. *Mobile Safari setting: Web Inspector*

2. In Safari on your computer, in the menu bar, click Safari ➤ Preferences ➤ Advanced and select the “Show Develop menu in menu bar” check box, as shown in Figure 2-15.

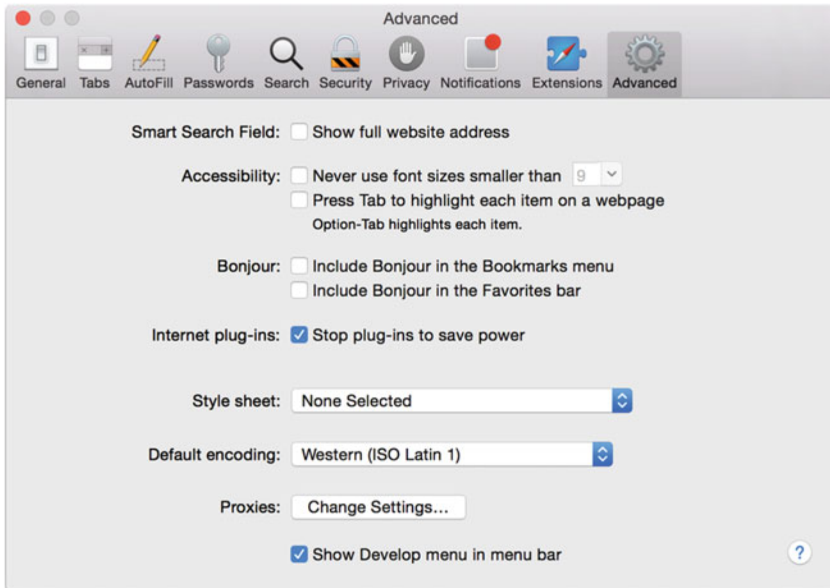


Figure 2-15. *Desktop Safari setting: Show Develop menu in menu bar*

3. Check whether you can see the Develop menu in the Safari menu bar (Figure 2-16).

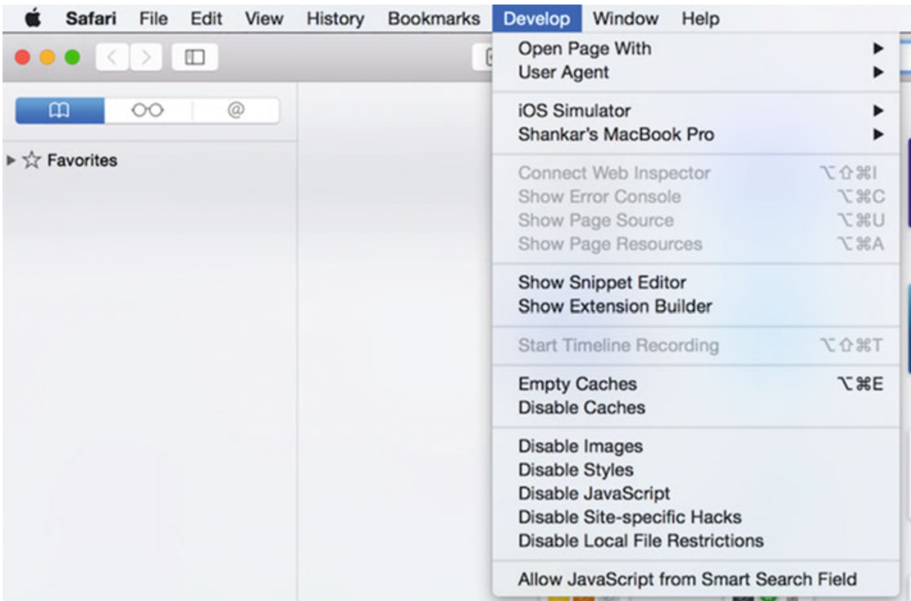


Figure 2-16. *Safari: Develop menu*

4. If you can see the Develop menu in the menu bar, check whether you see your iOS simulator or iPhone in the Develop menu (Figure 2-17).

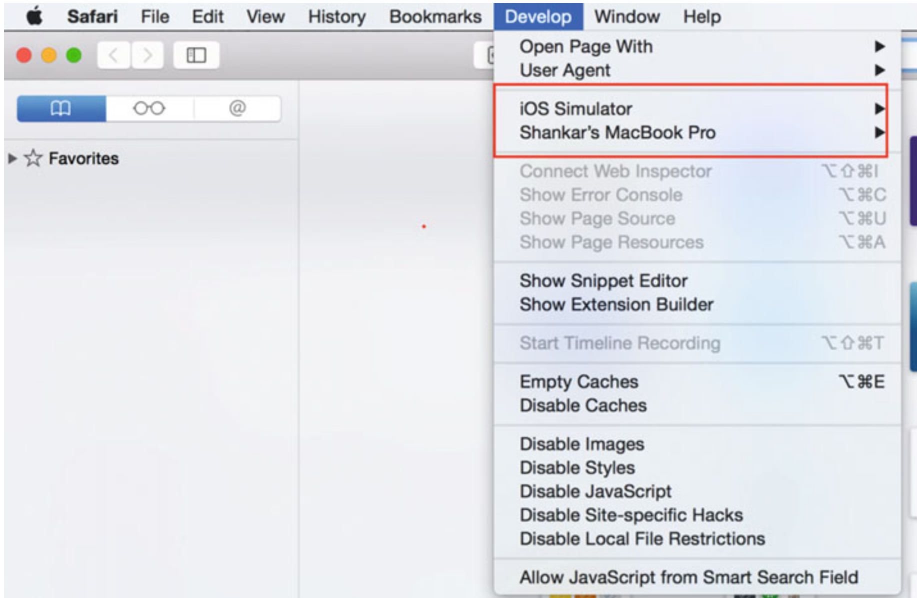


Figure 2-17. *Safari: iOS Simulator option*

5. Open Safari in the simulator and then open Google.com (Figure 2-18).

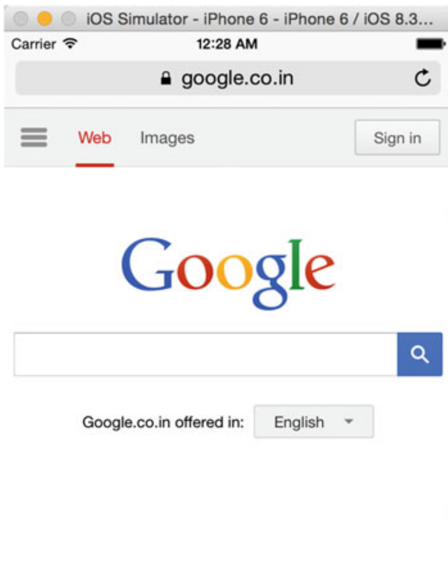


Figure 2-18. *Google.com on mobile Safari*

6. In Safari on your computer, select Develop ➤ iOS Simulator ➤ `www.google.com`, as shown in (Figure 2-19).

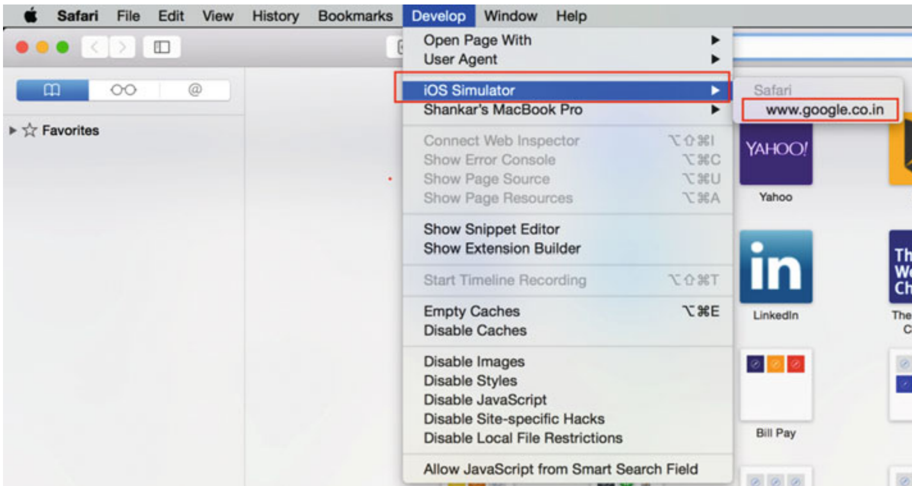


Figure 2-19. *Select the web site opened in the simulator in Safari on your computer*

Safari's Web Inspector will open, and you can use it to find elements.

7. Find the ID of an element using the Safari plug-in. Here is an example of the Google search page:

   a. Navigate to `https://www.google.com` on your mobile Safari browser.

   b. Open Web Inspector and click the Inspect button.

   c. Open the simulator and in mobile Safari click the element you want to find a locator for.

   d. See that the locator of that element is highlighted in Safari's Web Inspector.

You can now use the highlighted element property (Figure 2-20) in the Appium code.

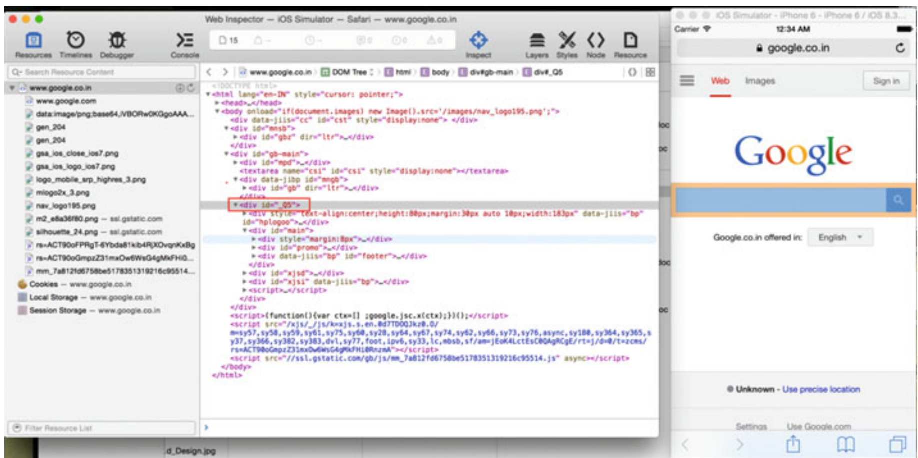


Figure 2-20. *Safari inspector for Google.com*

How It Works

The Develop menu option in the desktop version of Safari has an inspector for inspecting web elements, and the mobile version of Safari has a Web Inspector setting. When you use both of these settings in conjunction, you can use Web Inspector in the desktop version of Safari to inspect whichever web site is opened in mobile Safari. Here Safari is used as an example, but Safari's Web Inspector usage and UI are the same as the Firefox and Chrome inspectors.

2-7. Inspect Android Mobile Web Elements

Problem

You want to use the Chrome ADB plug-in to find Android mobile web elements.

Solution

You need to enable USB Debugging on the Android device so that it can be connected to a laptop.

1. Go to Settings ➤ About Phone and tap "Build number" seven times (Android 4.2 or above); then return to the previous screen and find "Developer options" (Figure 2-21).

Figure 2-21. *"Build number" item in "About phone" settings*

2. Tap "Developer options" and click On in the developer settings. (You will get an alert to allow the developer settings; just click the OK button.) Make sure the "USB debugging" option is checked (Figure 2-22).

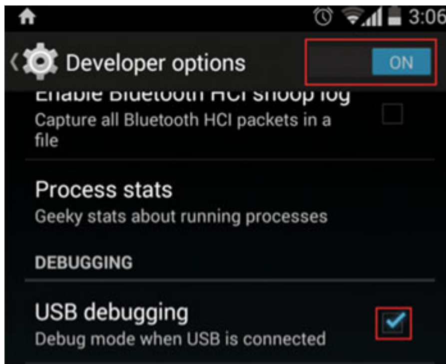


Figure 2-22. *"Developer options" settings and "USB debugging" option*

3. Connect your Android device to your computer (you should have installed the USB driver for your device). After connecting, you will get an alert on your device to allow USB debugging; just tap OK.

4. Download and install the Chrome ADB plug-in from `https://chrome.google.com/webstore/detail/adb/dpngiggdglpdnjdoaefidgiigpemgage?hl=en-GB`. Make sure you have installed Chrome version 32 or newer.

5. Open Chrome on your computer and click the ADB plug-in icon, which is in the top-right corner, and click View Inspection Targets (Figure 2-23).

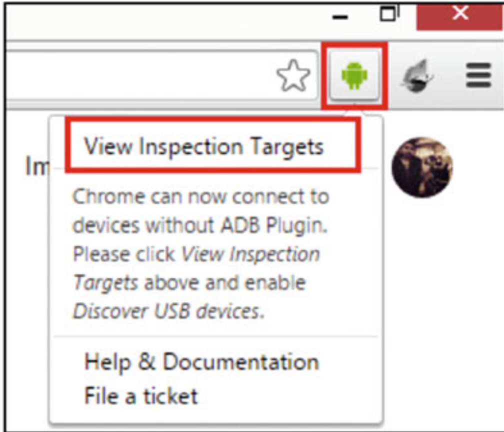


Figure 2-23. *Inspection targets in the Chrome ADB plug-in*

6. Open Chrome on your device and navigate to the desired URL (Google.com).

7. Go to `chrome://inspect/#devices`. This page will display all the connected devices along with open tabs and web views. Make sure "Discover USB devices" is selected. Now click the "inspect" link to open the developer tools (Figure 2-24).

Figure 2-24. *Discovering USB devices*

8. You will get the screen shown in Figure 2-25. Now click the screencast icon in the top-right corner to display your device screen. You are all set to find elements with the Chrome ADB plug-in.

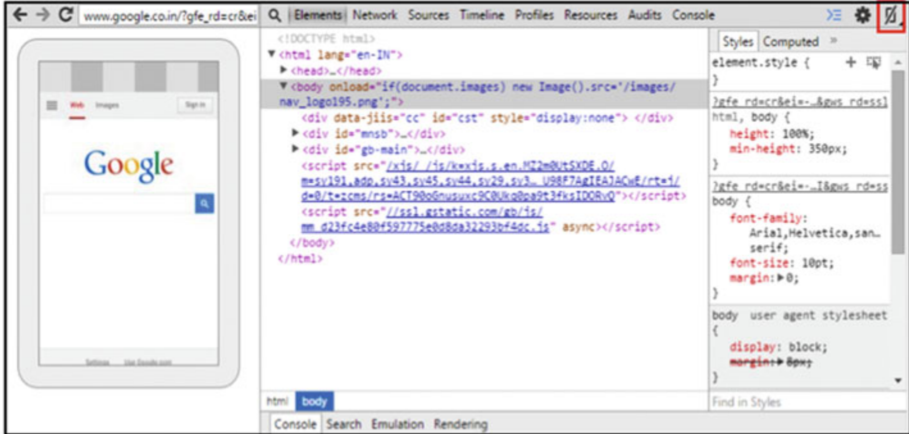


Figure 2-25. Screencast icon in Chrome ADB plug-in

9. Here you will find the ID of an element using the Chrome ADB plug-in remotely, with an example of the Google search page.

   a. Navigate to `https://www.google.com` on your mobile Chrome browser.

   b. Click the Inspect link from the ADB plug-in of your computer's Chrome browser.

   c. Click the inspect element icon and mouse over the search box.

The property of that element will be highlighted and can be used for Appium tests (Figure 2-26).

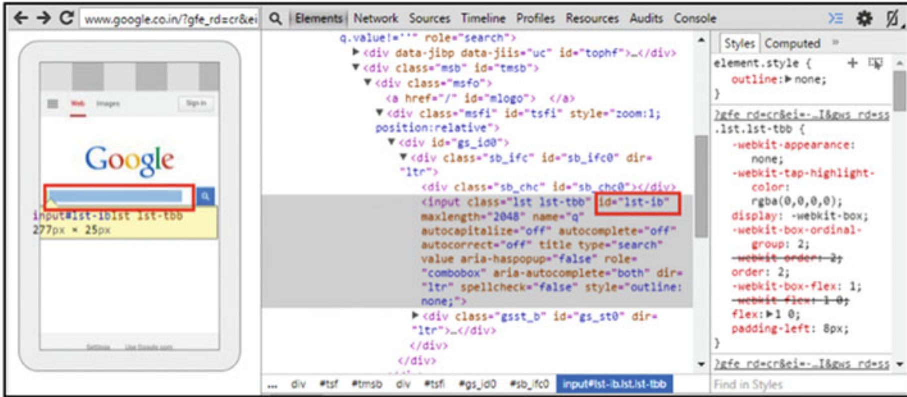


Figure 2-26. *ADB inspector for Google.com*

How It Works

USB debugging should be enabled on the device so that it is recognized by a computer as a connected device. The Chrome ADB plug-in allows you to view all the connected devices and web views. Select the device/web view and click Inspect to view mobile web elements and their properties.

CHAPTER 3

Automating Different Apps

In this chapter, you will learn to Automate:

- Native apps
- Mobile web apps
- Hybrid apps
- Real devices

In previous chapters, you learned how to set up Appium and how to find an element's properties, to be used in test cases. Now you know enough to start automating apps using Appium.

This chapter will cover different types of apps such as native, mobile web, and hybrid. First you will learn how to run test cases on emulators/simulators, and later you will learn to run them on real devices.

3-1. Native Apps

Problem

Native apps are perhaps the biggest reason why smartphones are so popular. Also, the majority of organizations start their mobile strategy with native apps. If you want to succeed in mobile automation, so you should know to automate a native app.

Note Appium's team maintains a separate repository for all apps that are used for sample test cases. You can download this repository from https://github.com/appium/sample-code. Once you download it, go to the `apps` folder and select the appropriate app for your test case. Apps for both Android and iOS are available. I have already included the sample apps in the `src/test/resources/apps` folder of the project you will use for this book (`AppiumRecipesBook`).

© Shankar Garg 2016
S. Garg, *Appium Recipes*, DOI 10.1007/978-1-4842-2418-2_3

Solution

You will automate a native app for both Android and iOS and perform some basic actions such as clicking and typing. These apps are demo apps developed by Appium's team and are good candidates to learn mobile automation.

Android App: ApiDemos-debug

Follow these steps:

1. In the `AppiumRecipesBook` project, in the `src/test/java` package, create a new class called `AppiumSampleTestCaseAndroid` with a `main()` function.
2. Add the following code in this class for the driver initialization and also for the implicit and explicit wait initialization:

```
WebDriver driver;
WebDriverWait wait;

// setting capabilities
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("platform", "ANDROID");
caps.setCapability("platformVersion", "5.0");
caps.setCapability("deviceName", "ANDROID");
caps.setCapability("browserName", "");

// relative path to apk file
final File classpathRoot = new
File(System.getProperty("user.dir"));
final File appDir = new File(classpathRoot, "src/test/
resources/apps/");
final File app = new File(appDir, "ApiDemos-debug.
apk");
caps.setCapability("app", app.getAbsolutePath());

// initializing driver object
driver = new AndroidDriver(new
URL("http://127.0.0.1:4723/wd/hub"), caps);

// initializing waits
driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);
wait = new WebDriverWait(driver, 10);
```

3. With the following code, you are performing the following actions on the Android app:

    a. Click Accessibility.

    b. Click Accessibility Node Querying.

    c. Click Back.

```
// Test Steps
// click on Accessibility link
wait.until(ExpectedConditions.presenceOfElementLocated(
MobileBy.AccessibilityId("Accessibility")));
driver.findElement(MobileBy.AccessibilityId("Accessi
bility")).click();

// click on 'Accessibility Node Querying' link
wait.until(ExpectedConditions.presenceOfElementLoc
ated(MobileBy.AccessibilityId("Accessibility Node
Querying")));
driver.findElement(MobileBy.AccessibilityId("Accessibility
Node Querying")).click();

// back
driver.navigate().back();

//close driver
driver.quit();
```

4. Run the Appium server on a terminal.

```
appium
```

5. Open the Genymotion console and run one Android emulator.

■ **Note** If you need information regarding how to set up Genymotion or the Android software development kit (SDK), please follow the instructions here:

https://shankargarg.wordpress.com/2016/02/25/setup-android-sdk-and-android-emulators/

https://shankargarg.wordpress.com/2016/02/25/setup-genymotion-android-emulators-on-mac-os/

---

6. Go to the program just written, right-click, and select Run as ➤ Java application.

The Appium server should receive the request, and the program should be executed appropriately (Figure 3-1).

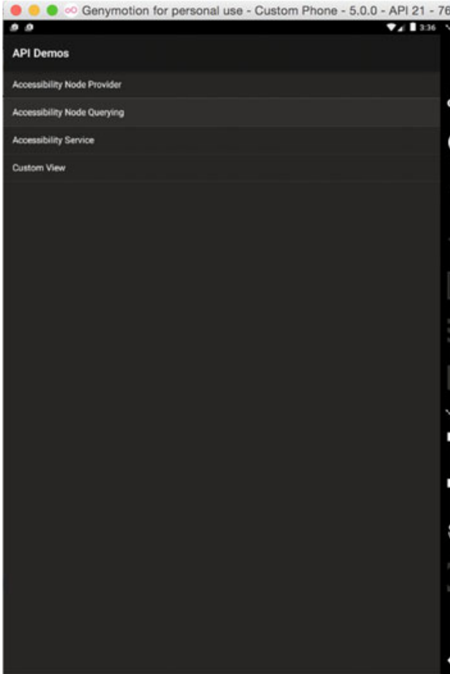


Figure 3-1. *Android sample test case*

iOS App: TestApp

Follow these steps:

1. In the `AppiumRecipesBook` project, in the `src/test/java` package, create a new class called `AppiumSampleTestCaseiOS` with a `main()` function.

2. Add the following code in this class for the driver initialization and also for the implicit and explicit wait initialization:

```
WebDriver driver;
WebDriverWait wait;

// setting capabilities
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("platform", "iOS");
caps.setCapability("platformVersion", "9.0");
caps.setCapability("deviceName", "iPhone 6");

// relative path to .app file
final File classpathRoot = new
File(System.getProperty("user.dir"));
final File appDir = new File(classpathRoot, "src/test/
resources/apps/");
final File app = new File(appDir, "TestApp.app");
caps.setCapability("app", app.getAbsolutePath());

// initializing driver object
driver = new IOSDriver(new URL("http://127.0.0.1:4723/
wd/hub"), caps);

// initializing waits
driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);
wait = new WebDriverWait(driver, 10);
```

3. With the following code, you are performing the following actions on an iOS app:

    a. Type AppiumBook in the first text box.

    b. Type First TC in the second text box.

```
// Test Steps
//enter data in first text box
wait.until(ExpectedConditions.presenceOfElementLocated(
MobileBy.AccessibilityId("TextField1")));
driver.findElement(MobileBy.AccessibilityId("TextFie
ld1")).sendKeys("AppiumBook");
```

```
//enter data in second text box
wait.until(ExpectedConditions.presenceOfElementLocated(
MobileBy.AccessibilityId("TextField2")));
driver.findElement(MobileBy.AccessibilityId("TextFie
ld2")).sendKeys("First TC");

//close driver
driver.quit();
```

4. Run the Appium server on a terminal.

```
appium
```

■ **Note** If you need information regarding how to set up iOS, please follow the steps mentioned here:

https://shankargarg.wordpress.com/2016/02/29/how-to-install-xcode-command-line-tools-and-ios-simulators-on-mac/

---

5. Go to the program just written, right-click, and select Run as ➤ Java application.

The Appium server should receive the request, and the program should be executed appropriately (Figure 3-2). Appium will open the iOS simulator.

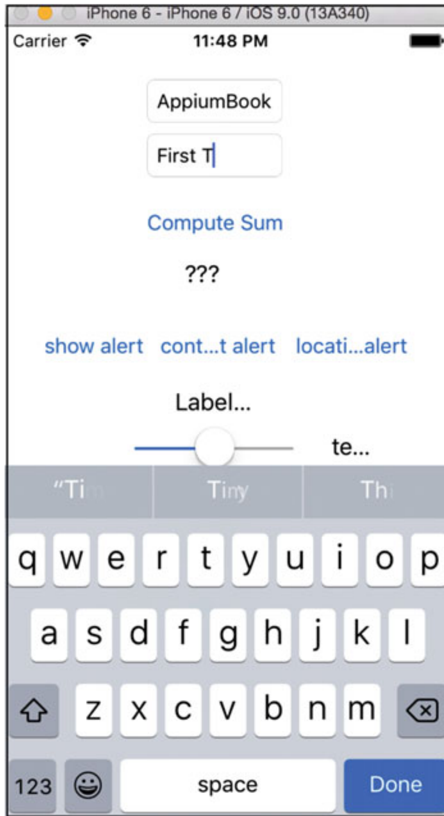


Figure 3-2. *iOS sample test case*

How It Works

Starting an Appium session for native apps depends on the capabilities set while starting the session. Capabilities such as `platform`, `platformversion`, and `deviceName` will decide the OS, but capabilities such as `browserName` and `app` will decide whether the session will be a native one.

■ **Note** For native sessions, `browserName` should be left blank, and the `app` capability should be the absolute local path or remote HTTP URL of the native app to be automated.

Once an Appium session is created for either Android or iOS, the same concepts as in Selenium are applied. You need to initialize explicit and implicit wait to enable Appium to wait for UI elements efficiently. Then you need to find elements with mobile automation locator strategies so you can appropriately interact with the elements.

3-2. Mobile Web Apps

Problem

Smartphones are the primary way most people connect to the Internet, and thus mobile web apps have become common in all organizations. All web sites that work on desktop browsers should work on mobile browsers as well. With the advent of development frameworks that allow creation of web sites for all form factors (such as desktop and mobile) with the same code, automating mobile web apps is a necessity that can't be overlooked.

Luckily, Appium automates the mobile web efficiently and without too much change. You want to understand how you can use Appium to automate mobile web apps.

Solution

To understand how to install Chrome and other Google Play store apps, please visit `https://shankargarg.wordpress.com/2016/08/04/install-google-play-store-and-chrome-on-genymotion-virtual-device/`.

Android

You will automate `https://github.com/` on Chrome on the Android emulator.

1. In the `AppiumRecipesBook` project, in the `src/test/java` package, create a new class called `AppiumSampleTestCaseAndroidWeb` with a `main()` function.

2. Add the following code in this class for the driver initialization and also for the implicit and explicit wait initialization:

```
//Declaring WebDriver variables
WebDriver driver;
WebDriverWait wait;

// setting capabilities
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("platform", "ANDROID");
caps.setCapability("platformVersion", "5.0");
caps.setCapability("deviceName", "ANDROID");
caps.setCapability("browserName", "chrome");
```

```
 // initializing driver object
 driver = new AndroidDriver(new
 URL("http://127.0.0.1:4723/wd/hub"), caps);

//initializing  waits
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
 wait = new WebDriverWait(driver, 10);
```

3. With the following code, you are performing the following actions on https://github.com/:

    a. Open https://github.com/.

    b. Click the Sign up for GitHub button.

    c. Click Create Account.

```
// Test Steps
//open github URL
driver.get("https://github.com/");

//click Signup
wait.until(ExpectedConditions.presenceOfElementLocated(
By.linkText("Sign up for GitHub")));
driver.findElement(By.linkText("Sign up for GitHub")).
click();

//click Create Account
wait.until(ExpectedConditions.presenceOfElementLocated(
MobileBy.id("signup_button")));
driver.findElement(MobileBy.id("signup_button")).
click();
//close driver
driver.quit();
```

4. Run the Appium server on a terminal.

```
appium
```

5. Go to the program just written, right-click, and select Run as ➤ Java application.

The Appium server should receive the request, and the program should be executed appropriately, as shown in Figure 3-3.

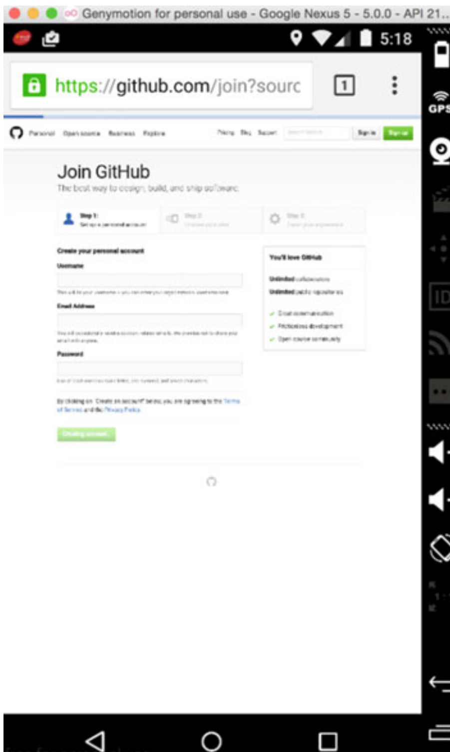


Figure 3-3. *Android test case on Chrome for* `https.//github.com/`

iOS

Follow these steps:

1. In the `AppiumRecipesBook` project, in the `src/test/java` package, create a new class called `AppiumSampleTestCaseiOSWeb` with a `main()` function.

2. Add the following code in this class for the driver initialization and also for the implicit and explicit wait initialization:

```
//Declaring WebDriver variables
        WebDriver driver;
        WebDriverWait wait;
```

```
// setting capabilities
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("platform", "iOS");
caps.setCapability("platformVersion", "9.0");
caps.setCapability("deviceName", "iPhone 6");
caps.setCapability("browserName", "safari");

// initializing driver object
driver = new IOSDriver(new URL("http://127.0.0.1:4723/wd/
hub"), caps);

// initializing waits
driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);
wait = new WebDriverWait(driver, 10);
```

3. With the following code, you are performing the following actions on `https://github.com/`:

   a. Open `https://github.com/`.

   b. Click the Sign up for GitHub button.

   c. Click Create Account.

```
// Test Steps
//open github URL
driver.get("https://github.com/");
// click Signup
wait.until(ExpectedConditions.presenceOfElementLocated(By.
linkText("Sign up for GitHub")));
driver.findElement(By.linkText("Sign up for GitHub")).click();

// click Create Account
wait.until(ExpectedConditions.presenceOfElementLocated(MobileBy.
id("signup_button")));
driver.findElement(MobileBy.id("signup_button")).click();

// close driver
driver.quit();
```

4. Run the Appium server on a terminal.

```
appium
```

5. Go to the program just written, right-click, and select Run as ➤ Java application.

The Appium server should receive the request, and the program should be executed appropriately (Figure 3-4). Appium will open the iOS simulator.

Figure 3-4. *iOS test case on Safari for https.//github.com/*

How It Works

For mobile web sessions, the `app` capability should not be set, and `browserName` should be the name of the mobile web browser to automate. Valid values for `browserName` are `Safari` for iOS and `Chrome`, `Chromium`, or `Browser` for Android. Using `Chrome` will open Chrome, and using `Browser` will open the default web browser installed on an Android device.

■ **Note** Here's an example of the beauty of Appium: the code for automating web apps is platform independent. The only difference in automating mobile web apps for iOS and Android is in the session creation part. This is why Appium is one of the most popular tools for mobile automation.

3-3. Hybrid Apps

Problem

A native app in which control passes from the native view to the web view is called a *hybrid app*. Although most organizations want to create a pure native app to gain better control and better access to user information, some parts of apps have to be mobile web such as a payment gateway page in an e-commerce app. To automate a native app fully, you want to learn how to automate hybrid apps.

Solution

You will automate a sample hybrid app for both Android and iOS, switch the context to the web view, and perform some basic actions such as clicking and typing on a web view.

Android

The demo app (the Selendroid sample app) can be downloaded from `http://selendroid.io/setup.html`. I've already added it to the `src/test/resources/apps` package for you.

1. In the `AppiumRecipesBook` project, in the `src/test/java` package, create a new class called `AppiumSampleTestCaseAndroidHybrid` with a `main()` function.

2. Add the following code in this class for the driver initialization and also for the implicit and explicit wait initialization:

```
// Declaring WebDriver variables
AndroidDriver<WebElement> driver;
WebDriverWait wait;

// setting capabilities
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("platform", "ANDROID");
caps.setCapability("platformVersion", "5.0");
caps.setCapability("deviceName", "ANDROID");
caps.setCapability("browserName", "");

// relative path to apk file
final File classpathRoot = new File(System.
getProperty("user.dir"));
final File appDir = new File(classpathRoot, "src/test/
resources/apps/");
final File app = new File(appDir, "selendroid-test-app.
apk");
caps.setCapability("app", app.getAbsolutePath());

// initializing driver object
driver = new AndroidDriver(new
URL("http://127.0.0.1:4723/wd/hub"), caps);

// initializing waits
driver.manage().timeouts().implicitlyWait(10, TimeUnit.
SECONDS);
wait = new WebDriverWait(driver, 10);
```

3. With the following code, you are performing the following actions on the sample app:

    a. Open the Selendroid sample app.

    b. Click the Chrome web view button.

    c. Switch to the web view.

    d. Enter Appium in text field on the web page.

    e. Click Submit.

```
// Test Steps
// click on Chrome icon to start web view
wait.until(ExpectedConditions.presenceOfElement
Located(MobileBy.id("io.selendroid.testapp:id/
buttonStartWebview")));
driver.findElement(MobileBy.id("io.selendroid.
testapp:id/buttonStartWebview")).click();

//Get all Contexts
Set<String> contexts = driver.getContextHandles();
for (String context : contexts) {

    //print Context name
    System.out.println(context);
    //switch to context containing web its name
    if (context.contains("WEB")) {
        driver.context(context);
    }
}

final WebElement inputField = driver.
findElement(By.id("name_input"));
inputField.sendKeys("Appium");
inputField.submit();

// close driver
driver.quit();
```

4. Run the Appium server on a terminal.

```
appium
```

5. Go to the program just written, right-click, and select Run as ➤ Java application.

The Appium server should receive the request, and the program should be executed appropriately, as shown in Figure 3-5.

Figure 3-5. Android test case for hybrid app

iOS

The demo app (the iOS sample app `WebViewApp`) has already been added to the `src/test/resources/apps` package.

1. In the `AppiumRecipesBook` project, in the `src/test/java` package, create a new class called `AppiumSampleTestCaseiOSHybrid` with a `main()` function.

2. Add the following code in this class for the driver initialization and also for the implicit and explicit wait initialization:

```
//Declaring WebDriver variables
WebDriver driver;
WebDriverWait wait;

// setting capabilities
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("platform", "iOS");
caps.setCapability("platformVersion", "9.0");
caps.setCapability("deviceName", "iPhone 6");
```

```
// relative path to .app file
final File classpathRoot = new
File(System.getProperty("user.dir"));
final File appDir = new File(classpathRoot, "src/test/
resources/apps/");
final File app = new File(appDir, "WebViewApp.app");
caps.setCapability("app", app.getAbsolutePath());

// initializing driver object
driver = new IOSDriver(new URL("http://127.0.0.1:4723/
wd/hub"), caps);

// initializing waits
driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);
wait = new WebDriverWait(driver, 10);
```

3. With the following code, you are performing the following actions on a sample hybrid app:

    a. Open the sample app.

    b. Enter the URL to open the web view.

    c. Switch to the web view.

    d. Click the Signup button.

    e. Click Create Account.

```
//Enter URL to open WebView
        driver.findElement(By.className
        ("UIATextField")).clear();
        driver.findElement(By.className("UIATextField")).
        sendKeys("https://github.com/");
        driver.findElement(MobileBy.AccessibilityId("Go")).
        click();

        //switch context:
        final Set<String> contextNames = ((AppiumDriver) driver).
        getContextHandles();
            for (final String contextName : contextNames) {
              System.out.println(contextName);
              if (contextName.contains("WEB")) {
                 ((AppiumDriver) driver).context(contextName);
```

```
                System.out.println("context switched to 
                webview");
            }
        }
    // click Signup
    wait.until(ExpectedConditions.presenceOfElementLocated(By
    .linkText("Sign up for GitHub")));
    driver.findElement(By.linkText("Sign up for GitHub")).
    click();

    // click Create Account
    wait.until(ExpectedConditions.presenceOfElementLocated(Mo
    bileBy.id("signup_button")));
    driver.findElement(MobileBy.id("signup_button")).click();

    // close driver
    driver.quit();
```

4. Run the Appium server on a terminal.

```
appium
```

5. Go to the program just written, right-click, and Run as ➤ Java application.

The Appium server should receive the request, and the program should be executed appropriately (Figures 3-6 and 3-7). Appium will open the iOS simulator.

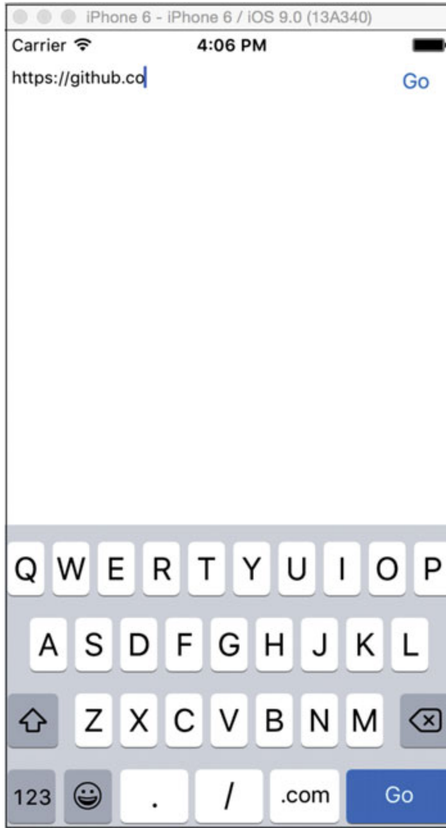


Figure 3-6. *iOS sample app to launch web view*

Figure 3-7. *iOS test case for hybrid app*

How It Works

For hybrid apps, the Appium session will be launched as a native app session only. So, there is no change in the capabilities while starting an Appium session. Once you reach a point when you need to interact with web view elements, then you need to switch the context to a web view context.

The context specifies how Appium interprets commands and which commands are available to the user. There are two types of contexts available in Appium.

- *Native*: This refers to native applications and to those parts of hybrid apps that are running native views. Commands in the native context execute against the device vendor's automation application programming interface (API) and interact directly with the device.

- *Web view*: This is part of a hybrid application that is inside a `UIAWebView` (for iOS) or `android.webkit.WebView` (for Android). In this context, the commands are used as standard `WebDriver` commands, giving access to elements through CSS selectors and other web-specific locators such as link text, and so on.

You use the context name as a string to switch between contexts. The native context will have the name `NATIVE_APP`, while the available web view contexts will have a name like `WEBVIEW_1` (for iOS) or `WEBVIEW_io.appium.android.apis` (for Android).

```
//Switch to specific web view
driver.context("contextName");
```

Once in the web view context, you can use Selenium commands to interact with a web application such as `driver.findElement(By.linkText("Sign up for GitHub")).click();`.

When you want to return to the native context, you use the same command as you used to get into the web view, but you ask to switch to the native context.

■ **Note** To identify elements in a hybrid view, refer to Chapter 2 to learn how to inspect Android mobile web elements and inspect Android mobile web elements.

3-4. Real Devices

Problem

Up to now you have learned how to automate native, web, and hybrid apps in an emulator for Android and in a simulator for iOS. Although emulators and simulators are almost as good as real devices, sometimes you want to test on an actual device.

Solution

Unlike traditional mobile automation tools, with Appium you don't need to make any substantial changes to your test cases to run them on real devices. You will automate native apps for both the Android and iOS platforms and run them on real Android and iOS devices, respectively.

Android

To run Android apps, Android devices should have developer mode enabled and should be connected to a computer and Android test case. You also need to enable USB debugging on the Android device for it to be used as a device for test case execution.

Follow these steps:

1. Go to Settings ➤ About Phone and tap "Build number" seven times, as shown in Figure 3-8.

Figure 3-8. *"Build number" setting on Android*

■ **Note** Tapping seven times is for the Google Nexus device. The number of times you need to tap will change from manufacturer to manufacturer, so do an Internet search for the number for your device if you're not sure.

---

2. You will get a success message that you're a developer.
3. Go back and select "Developer options" (Figure 3-9).

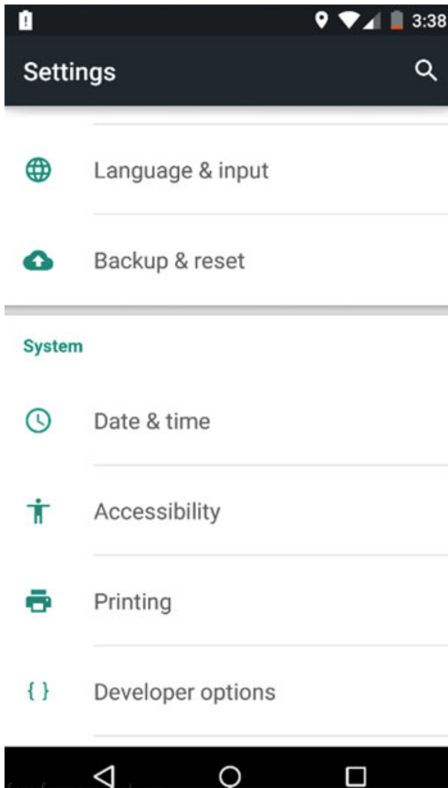


Figure 3-9. *"Developer options" setting on Android*

4. Enable "USB debugging" (Figure 3-10).

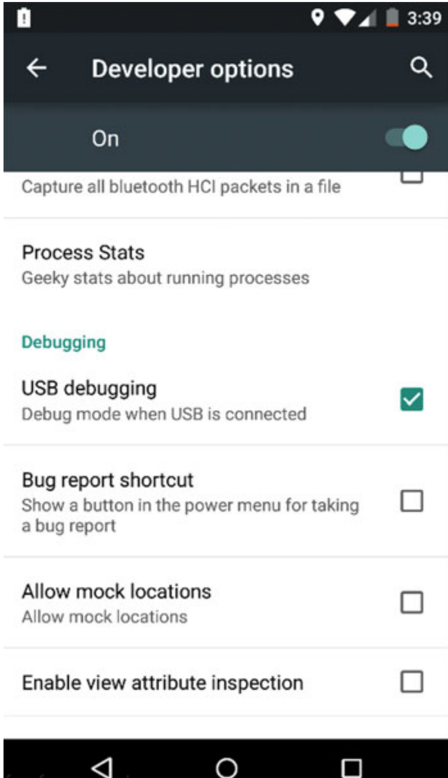


Figure 3-10. *"USB debugging" setting enabled on Android*

Now your device is ready to be connected to a computer that has the Android SDK installed. Let's enable your machine to connect to an Android device.

■ **Note** To know how to install the Android SDK on a computer, please refer to my blog at `https://shankargarg.wordpress.com/2016/02/25/setup-android-sdk-and-android-emulators/`.

---

5. Install a USB driver on your machine. There are many options available for this, but I prefer using PdaNet (`http://pdanet.co/`).

6. Please refer to the steps at `http://pdanet.co/help/mac.php` to install PdaNet. The steps are the same for Mac and Windows; only the downloaded file is different.

Once you have successfully installed PdaNet and enabled "USB debugging," let's check if the device is connected right to the computer.

7. Connect the device and run the following command on a terminal:

```
adb devices
```

The output of the previous command should look like Figure 3-11.

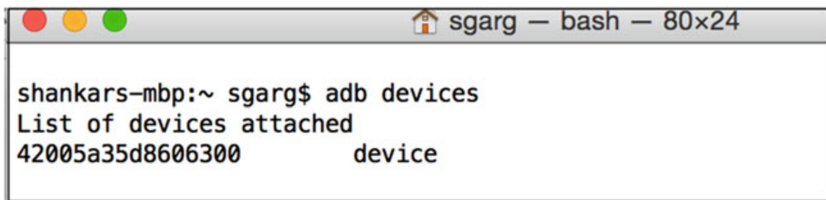


Figure 3-11. *Android real device as shown in a terminal*

8. If the previous command does not work or if the device is listed as inactive, you can stop the Android Debug Bridge (adb) server by using the command `adb kill-server` and then restart it by using the command `adb start-server`. Reconnect your device and execute `adb devices` again. Your device should be listed.

9. Run the Appium server on a terminal.

```
appium
```

10. Before running this test case, make sure that no Android emulator is running and that only one Android device is connected to the machine.

11. You are ready to execute the test case on a real device. Open the `AppiumSampleTestCaseAndroid` class, right-click, and select Run as ➤ Java application.

■ **Note** The device should be either unlocked or locked with a simple swipe lock. The Appium unlock app can't unlock four- or six-digit locks or pattern locks and will result in a test case failure.

---

12. Observe the Appium output and also the device screen. In a few seconds you should see the API-Demos app running.

iOS

To run a native app on a real iOS device, you need to sign the app for that device, connect the device to a computer, and add a device ID (UDID) to the test case to run it.

You need to create an Apple account so that you can create provisioning profiles to be used in installing apps on real devices.

1. Register at `https://developer.apple.com/programs/` and remember the credentials.
2. Go to `https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingProfiles/MaintainingProfiles.html#//apple_ref/doc/uid/TP40012582-CH30-SW24` and refer to the section "Creating Development Provisioning Profiles." Perform all the steps mentioned there.
3. Go to the section "Verifying and Removing Provisioning Profiles on Devices" to install the provisional profile created in the previous step on the real device (which will be used for test case execution).
4. Now you need to know the UDID of the real device.
   a. Using a USB cable, connect iOS to your Mac.
   b. Open Xcode and select Window ➤ Devices.
   c. Select "Connected device."
   d. Under Device Information, you will see an identifier like `46ba868066b970c7c6fe86bfe9d97c63abfeb565`. Now your device is ready to be used for the test case execution.
5. In the `AppiumRecipesBook` project, in the `src/test/java` package, create a new class called `AppiumSampleTestCaseiOSRD` with a `main()` function.
6. Add the following code in this class for the driver initialization and also for the implicit and explicit wait initialization:

```
//Declaring WebDriver variables
       WebDriver driver;
       WebDriverWait wait;

      // setting capabilities
       DesiredCapabilities caps = new DesiredCapabilities();
       caps.setCapability("platform", "iOS");
       caps.setCapability("platformVersion", "9.3.4");
       caps.setCapability("deviceName", "iPhone 6");
```

```
caps.setCapability("udid",
"46ba868066b970c7c6fe86bfe9d97c63abfeb363");

// relative path to .app file
final File classpathRoot = new
File(System.getProperty("user.dir"));
final File appDir = new File(classpathRoot, "src/test/
resources/apps/");
final File app = new File(appDir, "TestApp.app");
caps.setCapability("app", app.getAbsolutePath());

// initializing driver object
driver = new IOSDriver(new URL("http://127.0.0.1:4723/wd/
hub"), caps);

// initializing waits
driver.manage().timeouts().implicitlyWait(10,
TimeUnit.SECONDS);
 wait = new WebDriverWait(driver, 10);
```

■ **Note** Make sure to match the platform version exactly to the device connected.

---

7. With the following code, you are performing the following actions on the iOS app:
   a. Type **AppiumBook** in the first text box.
   b. Type **First TC** in the second text box.

```
// Test Steps
//enter data in first text box
wait.until(ExpectedConditions.presenceOfElementLocated
(MobileBy.AccessibilityId("TextField1")));
driver.findElement(MobileBy.AccessibilityId("TextField1")).
sendKeys("AppiumBook");

//enter data in second text box
wait.until(ExpectedConditions.presenceOfElementLocated
(MobileBy.AccessibilityId("TextField2")));
driver.findElement(MobileBy.AccessibilityId("TextField2")).
sendKeys("First TC");

//close driver
driver.quit();
```

8. Run the Appium server on a terminal.

```
appium
```

9. Go to the program just written, right-click, and select Run as ➤ Java application.

The Appium server should receive the request, and the program should be executed appropriately (Figure 3-12) on the connected device.

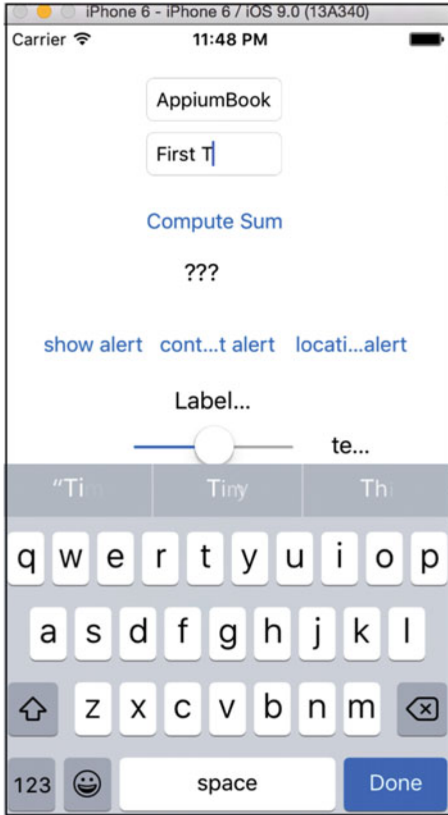


Figure 3-12. iOS sample test case on a real device

How It Works

The awesome thing about Appium is that you don't need to change a single line of code for real Android devices; you only need to add the UDID for real iOS devices.

All the other steps mentioned here are mostly to connect your device to a computer and are not related to Appium. So, once you have performed the steps and your device is connected, you are all set and never have to perform these steps again on the same device.

CHAPTER 4

Automating Mobility

In this chapter, you will learn to automate the following:

- Tap mobile elements
- Drag and drop elements
- Swipe and scroll
- Manage device orientation
- Install and uninstall native apps
- Lock and unlock devices
- Manage device network settings

In previous chapters, you learned to use Appium to automate different types of apps on different devices. To automate mobile apps, automating gestures (such as tapping, scrolling, swiping, and so on) is of utmost importance.

In earlier versions of Appium, you had to combine some generic functions to perform these complex functions, but in the latest versions, specific functions such as zooming, pinching, and so on, are available. These functions have their own syntaxes, which you'll learn in this chapter.

Some of the functions are available to only one platform, Android or iOS, and for others, their syntax will change depending on the underlying platform.

Note When functions have the same syntax and implementation for both Android and iOS, the recipes explain the concepts using the Android platform. Since Android can be executed on both Windows and Mac machines, it is useful for a larger audience. You can execute the same functions on iOS to gain better understanding.

© Shankar Garg 2016

S. Garg, *Appium Recipes*, DOI 10.1007/978-1-4842-2418-2_4

4-1. Tap Mobile Elements

Problem

For people familiar with web automation, clicking is a common and simple action, but in the mobile landscape, *tapping* is the action that replaces clicking. You want to know how to tap elements using Appium.

Solution

For this recipe, you will automate the process of tapping various menu options and buttons of an Android native app.

■ **Note** Tapping works the same for Android and iOS. To avoid redundancy, only an Android example is provided here.

Android

Follow these steps:

1. In the AppiumRecipesBook project, in the src/test/java package, create a new class called AppiumAndroidMobility with the following functions.
2. Add the following code in this class for the driver initialization and also for the implicit and explicit wait initialization:

```
public class AppiumAndroidMobility {

    // Declaring WebDriver variables
    static AndroidDriver<AndroidElement> driver;
    static WebDriverWait wait;

    static DesiredCapabilities caps = new
    DesiredCapabilities();

    public static void main(String[] args) throws
    InterruptedException, IOException {

       new AppiumAndroidMobility().
       settingCapsAndDriver();

new AppiumAndroidMobility().closeDriver();

    }
```

```
public void settingCapsAndDriver() throws
MalformedURLException {
       // setting capabilities
       caps.setCapability("platform", "ANDROID");
       caps.setCapability("platformVersion", "5.0");
       caps.setCapability("deviceName", "ANDROID");
       caps.setCapability("browserName", "");

       // relative path to apk file
       final File classpathRoot = new File(System.
       getProperty("user.dir"));
       final File appDir = new File(classpathRoot,
       "src/test/resources/apps/");
       final File app = new File(appDir, "ApiDemos-
       debug.apk");
       caps.setCapability("app", app.
       getAbsolutePath());

       // initializing driver object
       driver = new AndroidDriver<AndroidElement>(new
       URL("http://127.0.0.1:4723/wd/hub"), caps);

       // initializing waits
       driver.manage().timeouts().implicitlyWait(10,
       TimeUnit.SECONDS);
       wait = new WebDriverWait(driver, 10);
    }

    public void closeDriver() {
       // close driver
       driver.quit();
    }
}
```

3. With the following code, you are performing the following actions on an Android app:

    a. Click the Views option.

    b. Click the Buttons option.

    c. Tap the OFF button.

    d. Print the text of the OFF button that has changed to ON now.

```
public void taponElement() {
       // Start - Ch.4-R.1
       // Tap
       driver.findElement(MobileBy.
       AccessibilityId("Views")).click();
       driver.findElement(MobileBy.
       AccessibilityId("Buttons")).click();

       Point point = driver.findElementById("io.appium.
       android.apis:id/button_toggle").getLocation();
       driver.tap(1, point.x + 20, point.y + 30, 1000);
       System.out.println(driver.findElementById
       ("io.appium.android.apis:id/button_toggle").
       getText());
       // End - Ch.4-R.1

    }
```

4. Call the function created in the previous step in the main function using the following code:

```
new AppiumAndroidMobility().taponElement();
```

5. Run the Appium server on a terminal, run an Android emulator, and execute the program as explained in the previous chapters.

The Appium server should receive the request, and the program should be executed appropriately (Figure 4-1).

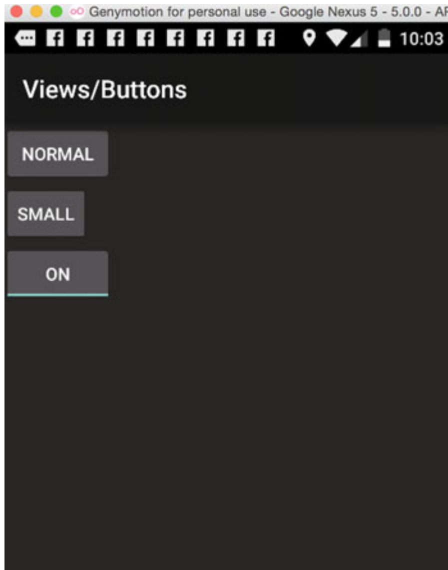


Figure 4-1. *Tap function for Android app*

How It Works

Tapping is a method in the `TouchAction` class and is used to perform click operations on mobile elements. The tap method can be used with two different options.

- `tap(fingers, element, duration)`: Here the first argument, `finger`, is how many fingers you want to use for tapping, such as 1 or 2. The second argument, `element`, is the mobile element on which to tap (this is the result of a statement like `driver.findElement()`). The third argument, `duration`, is the time to perform a tap; for instance, 1000 or 2000 ms means 1 or 2 seconds.
- `tap(fingers, x, y, duration)`: Here the first argument, `finger`, is how many fingers you want to use for tapping, such as 1 or 2. The second and third arguments, `x` and `y`, are absolute coordinates at which the tap will be performed. The fourth argument, `duration`, is the time to perform a tap. For instance, 1000 or 2000 ms means 1 or 2 seconds.

4-2. Drag and Drop Elements

Problem

You want to select an element, drag it from its original position, and drop it on some other position/element. (This is a common task in gaming apps.)

Solution

For this recipe, you will automate dragging an element from its original location and dropping it on a target location.

■ **Note** Dragging and dropping works the same for Android and iOS. To avoid redundancy, only an Android example is provided here.

Android

Follow these steps:

1. In the `AppiumAndroidMobility` class, comment the code written for calling the tap function.
2. With the following code, you are performing the following actions on an Android app:
   a. Click the Views option.
   b. Click the Drag and Drop option.
   c. Hold and drag Dot 1.
   d. Drop Dot 1 on Dot 3.
   e. Print the text that has changed after dragging and dropping.

```
public void dragDrop() {
       // Start - Ch.4-R.2
       // Drag and Drop
       // Open an activity directly
       driver.startActivity("io.appium.android.
       apis", ".view.DragAndDropDemo");

       WebElement dragDot1 = driver.findElement(By.
       id("io.appium.android.apis:id/drag_dot_1"));
       WebElement dragDot3 = driver.findElement(By.
       id("io.appium.android.apis:id/drag_dot_3"));
```

```
        // this text should be empty before Drag-Drop
        WebElement dragText = driver.findElement(By.
        id("io.appium.android.apis:id/drag_text"));
        System.out.println(dragText.getText());

        // perform Drag and Drop
        TouchAction dragNDrop = new
        TouchAction(driver).longPress(dragDot1).
        moveTo(dragDot3).release().perform();

        // Text representing Drag-Drop is successful
        System.out.println((dragText.getText()));
        // End - Ch.4-R.2
    }
```

3. Call the function created in the previous step in the main function using the following code:

```
new AppiumAndroidMobility().dragDrop();
```

4. Run the Appium server on a terminal, run an Android emulator, and execute the program as explained in the previous chapters.

The Appium server should receive the request, and the program should be executed appropriately (Figure 4-2).

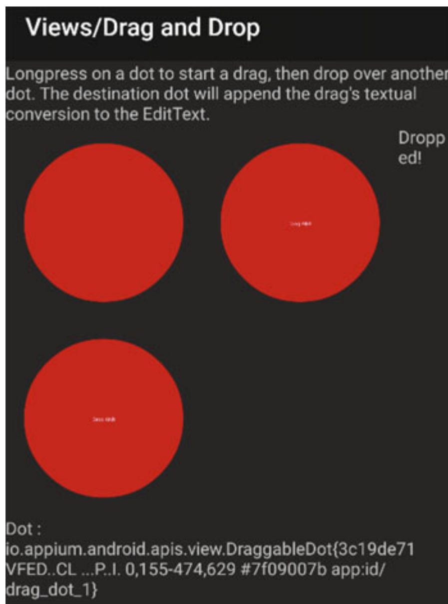


***Figure 4-2.** Dragging and dropping on an Android app*

How It Works

Dragging and dropping are complex actions performed by combining various simple methods available in the `TouchAction` class. These methods include the following:

- `longPress()` is to tap an element for a long duration.
- `moveTo` is to move the tapped element to another location.
- `release()` and `perform()` are part of a concept called *chaining of actions* where simple elements are chained one after another. `release()` chains the methods only locally, and nothing is sent to the Appium server to execute. Once the `perform()` method is executed, then only all chained methods are sent to the Appium server to be executed.

■ **Note** Using the concept of chaining simple actions, more complex actions can be automated easily, such as multitouch actions.

4-3. Swipe and Scroll

Problem

Swiping and scrolling are probably the most widely used mobility features, and this has made mobile usage very user friendly. You want to learn how to automate swiping and scrolling in mobile apps.

Solution

For this recipe, you will automate swiping on the screen (vertical and horizontal) on an Android app and also scrolling on a web element such as Scroller in an iOS app.

Android

Follow these steps:

1. In the `AppiumAndroidMobility` class, comment the code written for calling the drag-and-drop function.
2. With the following code, you are performing the following actions on an Android app:
   a. Click the Views option.
   b. Scroll up on the screen.

c. Print the text for the first element with accessibility ID `android:id/text1`.

d. Scroll down on the screen.

e. Print the text for the first element with accessibility ID `android:id/text1`.

```
public void swipeVertical() {
       // Start - Ch.4-R.3
       // vertical swipe
       driver.findElementByAccessibilityId("Vie
       ws").click();
       AndroidElement listView = driver.
       findElementByClassName("android.widget.
       ListView");
       MobileElement textView = driver.
       findElementById("android:id/text1");

       String originalText = textView.getText();

       listView.swipe(SwipeElementDirection.UP, 20,
       15, 1000);
       System.out.println(textView.getText());

       listView.swipe(SwipeElementDirection.DOWN,
       20, 15, 1000);
       System.out.println(textView.getText());
       // End - Ch.4-R.3

    }
```

3. Call the function created in the previous step in the main function using the following code:

```
new AppiumAndroidMobility().swipeVertical();
```

4. Run the Appium server on a terminal, run an Android emulator, and execute the program as explained in the previous chapters.

   The Appium server should receive the request, and the program should be executed appropriately.

5. Comment the code written in step 2.

6. With the following code, you are performing the following actions on the Android app:

    a. Click the Views option.

    b. Click the Gallery option.

    c. Click the Photo option.

    d. Scroll left on the screen.

    e. Scroll right on the screen.

```
public void swipeHorizontal() {
        // Start - Ch.4-R.3
        // horizontal swipe
        driver.findElementByAccessibilityId("Views").
        click();
        driver.findElementByAccessibilityId
        ("Gallery").click();
        driver.findElementByAccessibilityId("1.
        Photos").click();

        AndroidElement gallery = driver.
        findElementById("io.appium.android.apis:id/
        gallery");
        int originalImageCount = gallery.findElements
        ByClassName("android.widget.ImageView").size();

        gallery.swipe(SwipeElementDirection.LEFT, 5,
        5, 2000);
        System.out.println(gallery.findElementsBy
        ClassName("android.widget.ImageView").size());

        gallery.swipe(SwipeElementDirection.RIGHT,
        5, 5, 2000);
        System.out.println(gallery.findElementsByClass
        Name("android.widget.ImageView").size());
        // End - Ch.4-R.3

    }
```

7. Call the function created in the previous step in the main function using the following code:

```
new AppiumAndroidMobility().swipeHorizontal();
```

8. Run the Appium server on a terminal, run an Android emulator, and execute the program as explained in the previous chapters.

The Appium server should receive the request, and the program should be executed appropriately (Figure 4-3).

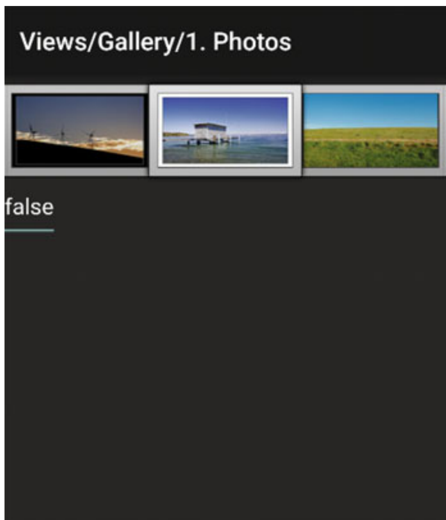


Figure 4-3. *Horizontal swiping on Android app*

iOS

Follow these steps:

1. In the AppiumRecipesBook project, in the src/test/java package, create a new class called AppiumIOSMobility with the following functions.

www.allitebooks.com

2. Add the following code in this class for the driver initialization and also for the implicit and explicit wait initialization:

```
public class AppiumIOSMobility {
    // Declaring WebDriver variables
    static IOSDriver<IOSElement> driver;
    static WebDriverWait wait;

    static DesiredCapabilities caps = new
DesiredCapabilities();

    public static void main(String[] args) throws
InterruptedException, IOException {

       new AppiumIOSMobility().settingCapsAndDriver();

       new AppiumIOSMobility().closeDriver();

    }

    public void settingCapsAndDriver() throws
MalformedURLException {
       // setting capabilities
       caps.setCapability("platform", "iOS");
       caps.setCapability("platformVersion", "9.2");
       caps.setCapability("deviceName", "iPhone 6");

       // relative path to .app file
       final File classpathRoot = new File(System.
       getProperty("user.dir"));
       final File appDir = new File(classpathRoot,
       "src/test/resources/apps/");
       final File app = new File(appDir, "TestApp.
       app");
       caps.setCapability("app", app.
       getAbsolutePath());

       // initializing driver object
       driver = new IOSDriver<IOSElement>(new
       URL("http://127.0.0.1:4723/wd/hub"), caps);

       // initializing waits
       driver.manage().timeouts().implicitlyWait(20,
       TimeUnit.SECONDS);
       wait = new WebDriverWait(driver, 20);
    }

    public void closeDriver() {
       // close driver
       driver.quit();
    }
}
```

3. With the following code, you are performing the following actions on the iOS app:

    a. Swipe the slider to the leftmost position.

    b. Swipe the slider to the rightmost position.

```
public void swipeiOS() {
        // Start - Ch.4-R.3
        // Horizontal Swipe
        MobileElement slider = driver.findElementByC
        lassName("UIASlider");

        // Scroll left
        slider.swipe(SwipeElementDirection.LEFT,
        slider.getSize().getWidth() / 2, 0, 3000);
        System.out.println(slider.
        getAttribute("value"));

        // Scroll Right
        slider.swipe(SwipeElementDirection.RIGHT, 2,
        0, 3000);
        System.out.println(slider.
        getAttribute("value"));
        // End - Ch.4-R.3
    }
```

4. Call the function created in the previous step in the main function using the following code:

```
new AppiumIOSMobility().swipeiOS();
```

5. Run the Appium server on a terminal and execute the program as explained in the previous chapters.

The Appium server should receive the request, and the program should be executed appropriately (Figure 4-4).

Figure 4-4. Slider scrolled to the rightmost position

■ **Note** The remaining recipes of the chapter will use `AppiumIOSMobility`. Each time, we will comment out the code previously written and write new functions. This way, at the end of this chapter you will have a class that has all the functions discussed in this chapter that is compact and useful.

How It Works

The `Swipe()` function is used for both horizontal and vertical swiping. The syntax for this function is as follows:

```
mobileElement.swipe(direction, offsetFromStartBorder, offsetFromEndBorder,
duration)
```

The following are the attributes in this function:

- `mobileElement` is the element on which the swipe will be performed, provided swiping is possible on this element.
- `direction` is an `ENUM`, which takes values such as `LEFT`, `RIGHT`, `UP`, and `DOWN` to set the direction of swipe.
- `offsetFromStartBorder` and `offsetFromEndBorder` are the offsets from the border of the element used for swiping. These will set the scope of the swipe.
- `duration` is the time in miliseconds to be taken for swiping.

If the element is across multiple screens, then swiping can be used for swiping across screens. For instance, in Android, if an element is small like a slider, swiping can be used to set the location of that slider like in the iOS example.

4-4. Manage Device Orientation

Problem

One convenience of using smartphones and tablets is that when you hold the device either horizontally or vertically, the mobile app will adjust to the new viewport size. In the beginning of the mobile app development era, most defects were discovered because of an orientation change, so it is important to run test cases on orientation change to make sure that the app works fine when users change the orientation. You want to know how to change the orientation using Appium.

Solution

For this recipe, you will automate an orientation change from portrait to landscape, and vice versa.

■ **Note** Orientation works the same for Android and iOS. To avoid redundancy, only an Android example is provided here.

Android

Follow these steps:

1. In the `AppiumAndroidMobility` class, comment the code written for calling the `Swipe()` function.

2. With the following code, you are performing the following actions on the Android app:

    a. Printing the current orientation, which is portrait

    b. Changing the orientation to landscape

    c. Printing the current orientation, which is landscape

    d. Changing the orientation back to portrait

```
public void changeOrientation() {
        // Start - Ch.4-R.4
        // Orientation
        // print current orientation
        System.out.println(driver.getOrientation());
        // change orientation to LANDSCAPE
        driver.rotate(ScreenOrientation.LANDSCAPE);

        // print current orientation
        System.out.println(driver.getOrientation());
        // change orientation to PORTRAIT
        driver.rotate(ScreenOrientation.PORTRAIT);
        // End - Ch.4-R.4
    }
```

3. Call the function created in the previous step in the main function using the following code:

```
new AppiumAndroidMobility().changeOrientation();
```

4. Run the Appium server on a terminal, run an Android emulator, and execute the program as explained in the previous chapters.

The Appium server should receive the request, and the program should be executed appropriately (Figure 4-5).

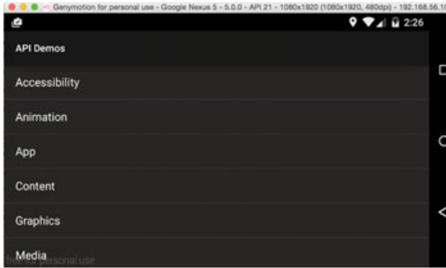


Figure 4-5. *Android app in landscape mode*

How It Works

The `Rotate()` function is used to change the existing orientation of a device, be it Android or iOS. The syntax for `Rotate()` is as follows:

```
driver.rotate(ScreenOrientation.'ORIENTATION')
```

Here `'ORIENTATION'` can be `LANDSCAPE` or `PORTRAIT` depending on the orientation you want. The `driver.getOrientation()` function is used to get the existing orientation of the device.

4-5. Install and Uninstall Native Apps

Problem

Installing, upgrading, and deleting applications can be tricky because these tasks require a lot of changes to the memory and cache on a device. So, testing scenarios related to these steps are important for covering edge scenarios in your test strategy.

Another important step is closing the app in between the test case and launching the app again. You want to know how to automate all these steps.

Solution

For this recipe, you will automate launching, closing, installing, and removing an app from an Android device.

■ **Note** At the time of this writing, for iOS functions `launchApp()` and `closeApp()` work fine, and the syntax is the same as Android, but functions such as `installApp()` and `removeApp()` are yet to be implemented for iOS in `java-client 4.0.0`.

Android

Follow these steps:

1. In the `AppiumAndroidMobility` class, comment the code written for calling the orientation change.
2. With the following code, you are performing the following actions on an Android app:
    a. Checking whether the app is launched
    b. Closing the app
    c. Launching the app again
    d. Checking whether the app is launched
    e. Checking whether app is installed
    f. Removing the app from the device
    g. Installing the app again
    h. Checking whether the app is installed

```
public void appLaunchClose() {
       // Start - Ch.4-R.5
       // App launch and Close

       // confirm if app is launched: - activity
       name should be from app
       System.out.println("Current Activity before
       Close: " + driver.currentActivity());

       // close the app
       driver.closeApp();

       // launch the app again
       driver.launchApp();
       // confirm if app is launched again: -
       activity name should be from app
       System.out.println("Current Activity after
       launch: " + driver.currentActivity());

       // App Installation
       // check if app is installed
       System.out.println("app installed before
       remove: " + driver.isAppInstalled
       ("io.appium.android.apis"));
```

```
    // remove app
    driver.removeApp("io.appium.android.apis");
    // check app is not installed now
    System.out.println("app installed after remove:
    " + driver.isAppInstalled("io.appium.android.
    apis"));

    // install app again
    // relative path to apk file
    final File classpathRoot = new File(System.
    getProperty("user.dir"));
    final File appDir = new File(classpathRoot,
    "src/test/resources/apps/");
    final File app = new File(appDir, "ApiDemos-
    debug.apk");
    driver.installApp(app.getAbsolutePath());

    // check if app is installed back
    System.out.println("app installed after install:
    " + driver.isAppInstalled("io.appium.android.
    apis"));
     // End - Ch.4-R.5
}
```

3. Call the function created in the previous step in the main function using the following code:

   ```
   new AppiumAndroidMobility().appLaunchClose();
   ```

4. Run the Appium server on a terminal, run an Android emulator, and execute the program as explained in the previous chapters.

The Appium server should receive the request. The program should be executed appropriately, and the output should look like Figure 4-6.

```
Problems  @ Javadoc  Declaration  Search  Console
<terminated> AppiumAndroidMobility [Java Application] /Library/Java/JavaVirt
Current Activity before Close: .ApiDemos
Current Activity after launch: .ApiDemos
app installed before remove: true
app installed after remove: false
app installed after install: true
```

Figure 4-6. *Android test case execution result for app launch, close, install, and remove*

How It Works

The function used to close the launched app is `driver.closeApp()`, and the function used to launch the app that is already installed is `driver.launchApp()`.

The function used to delete the installed app is `driver.removeapp()`, and the function used to install the app is `driver.installApp()`.

These functions are used extensively in scenarios such as deleting the existing version of app, upgrading the app, and then verifying the behavior.

4-6. Lock and Unlock Devices

Problem

You want to lock and unlock a device when testing an app's behavior, as well as in between the test cases.

Solution

For this recipe, you will automate locking and unlocking an Android device.

■ **Note** At the time of this writing, the lock and unlock functions work only with Android and not for iOS.

Android

Follow these steps:

1. In the `AppiumAndroidMobility` class, comment the code written for calling the app installation.

2. With the following code, you are performing the following actions on the sample app:

    a. Locking the device

    b. Checking the lock status

    c. Unlocking the device

    d. Checking the lock status

```
public void lockUnlock() {
       // Start - Ch.4-R.6
       // lock device:
       driver.lockDevice();
       System.out.println("After lock is device locked:
       " + driver.isLocked());

       driver.unlockDevice();
       System.out.println("After unlock is device
       locked: " + driver.isLocked());
       // End - Ch.4-R.6
   }
```

3. Call the function created in the previous step in the main function using the following code:

```
new AppiumAndroidMobility().lockUnlock();
```

4. Run the Appium server on a terminal, run an Android emulator, and execute the program as explained in the previous chapters.

The Appium server should receive the request, the program should be executed appropriately, and the output should look like Figure 4-7.

Problems @ Javadoc Declaration Search Console

<terminated> AppiumAndroidMobility [Java Application] /Library/Java/JavaVirt

```
After lock is device locked: true
After unlock is device locked: false
```

Figure 4-7. *Android test case execution result for device lock and unlock*

How It Works

`driver.lockDevice()` is used to lock the device, and `driver.unlockDevice()` is for unlocking the device. `driver.isLocked()` is used to check the status of the lock. Appium uses the unlock Android app to perform this operation.

4-7. Manage Device Network Settings

Problem

You want to test an app in different network settings such as no data mode or airplane mode to make sure that the app does not crash and that, when you switch back to Wi-Fi or data, the app resumes where it left off.

Solution

For this recipe, you will automate changing the network settings to data, Wi-Fi, airplane mode, and no data on an Android device.

■ **Note** At the time of this writing, the `setConnection` functions work only with Android and not for iOS.

Android

Follow these steps:

1. In the `AppiumAndroidMobility` class, comment the code written for calling the lock device.
2. With the following code, you are performing the following actions on the sample app:
    a. Set the network to `ALL`.
    b. Set the network to `AIRPLANE`.
    c. Set the network to `NONE`.
    d. Set the network to `WIFI`.

```
public void NetworkSettings() {

        // Start - Ch.4-R.7
        // Network
        driver.setConnection(Connection.ALL);
        System.out.println(driver.getConnection());

        driver.setConnection(Connection.AIRPLANE);
        System.out.println(driver.getConnection());

        driver.setConnection(Connection.NONE);
        System.out.println(driver.getConnection());

        driver.setConnection(Connection.WIFI);
        System.out.println(driver.getConnection());
        // End - Ch.4-R.7
    }
```

3. Call the function created in the previous step in the main function using the following code:

```
new AppiumAndroidMobility().NetworkSettings();
```

4. Run the Appium server on a terminal, run an Android emulator, and execute the program as explained in the previous chapters.

The Appium server should receive the request, the program should be executed appropriately, and the output should look like Figure 4-8.

```
Problems  @ Javadoc  Declaration  Search  Console
<terminated> AppiumAndroidMobility [Java Application] /Library/Java/JavaVirt
ALL
AIRPLANE
DATA
ALL
```

Figure 4-8. *Android test case execution result for network setting*

How It Works

`driver.SetConnection()` is used to set different network settings for Android devices. The syntax is as follows: `driver.setConnection(Connection.'NetworkSetting')`.

Here `NetworkSetting` is an `ENUM`, which could have values such as `ALL` for both cellular and Wi-Fi, `AIRPLANE` for airplane mode, `WIFI` for only Wi-Fi, and `NONE` for no network at all.

CHAPTER 5

Creating Automation Frameworks Using Appium

In this chapter, you will learn following:

- Create an automation framework with Appium, Maven, and TestNG
- Create a behavior-driven development (BDD) framework with Appium, Cucumber, and the page object model
- Conduct continuous automated testing with Appium, Git, and Jenkins

In previous chapters, you learned to use Appium to automate different apps and automate mobile-specific functions such as tapping, scrolling, swiping, and so on.

Appium's one and only functionality is to automate mobile platforms and mobile-specific functions. But for automation testing this is not sufficient. An automation framework should have different types of reporting, should integrate with continuous integration (CI)/continuous development (CD) tools, and should do much more. That's why you need to integrate Appium with other tools to create robust automation frameworks.

The following are some expectations of automation frameworks: integration with test runner and reporting tools such as TestNG and JUnit, BDD integration with Cucumber, and integration with CICD tools such as Jenkins. In this chapter, you'll understand how to integrate Appium with each of these tools.

5-1. Create an Automation Framework with Appium, Maven, and TestNG

Problem

For a robust automation framework, you need to integrate Appium with Maven for its dependency and life-cycle management capabilities and with TestNG for its capability to tag functions as test cases, to create HTML reports, to manage test cases, and so on. In this recipe, you want to know how to integrate Appium with Maven and TestNG.

© Shankar Garg 2016
S. Garg, *Appium Recipes*, DOI 10.1007/978-1-4842-2418-2_5

Solution

You will create an automation framework with Appium, Maven, and TestNG and write one sample test case for an Android app.

1. Install the Eclipse-TestNG plug-in by following the steps at `https://shankargarg.wordpress.com/2016/09/01/integrate-eclipse-and-testng/`.

2. Install the Eclipse-Maven plug-in by following the steps at `https://shankargarg.wordpress.com/2016/09/01/integrate-eclipse-and-maven/`.

3. Create a new project in Eclipse by following these steps: click New ➤ Other ➤ Maven ➤ Maven Project ➤ Next (Figure 5-1).

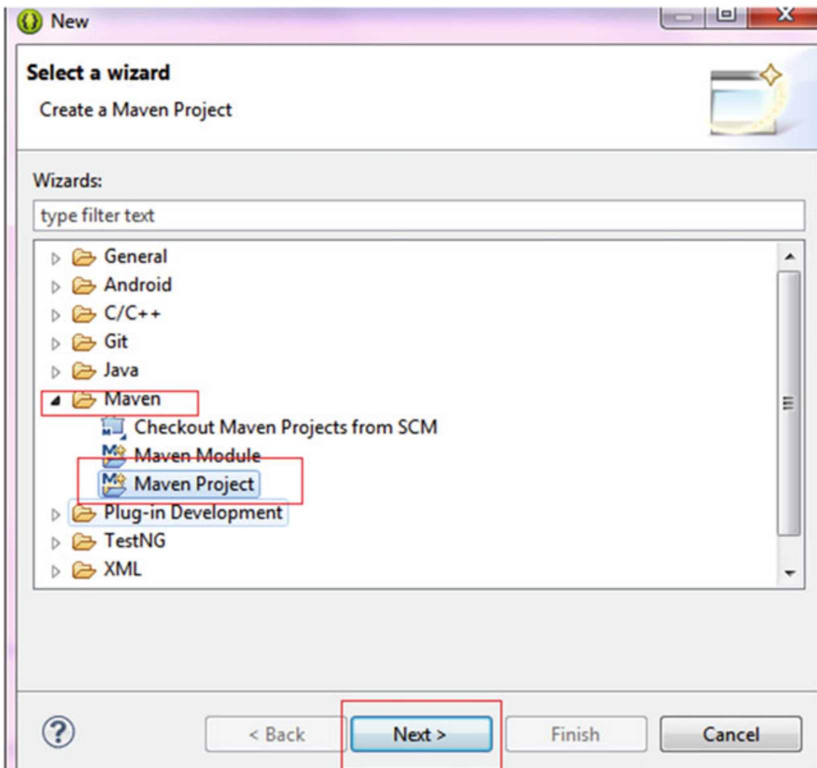


Figure 5-1. *Creating a new Maven project*

4. Select a simple project and keep the default workspace location (Figure 5-2).

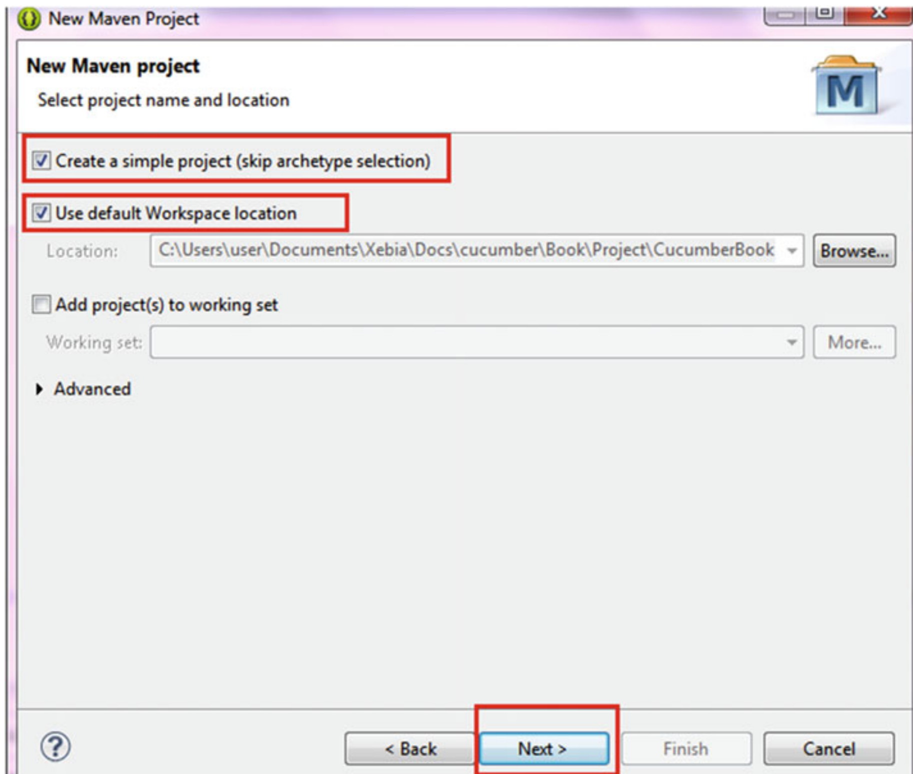


Figure 5-2. *Maven project creation wizard*

5. Provide details such as the artifact ID, group ID, name, and description. Then click Finish (Figure 5-3).

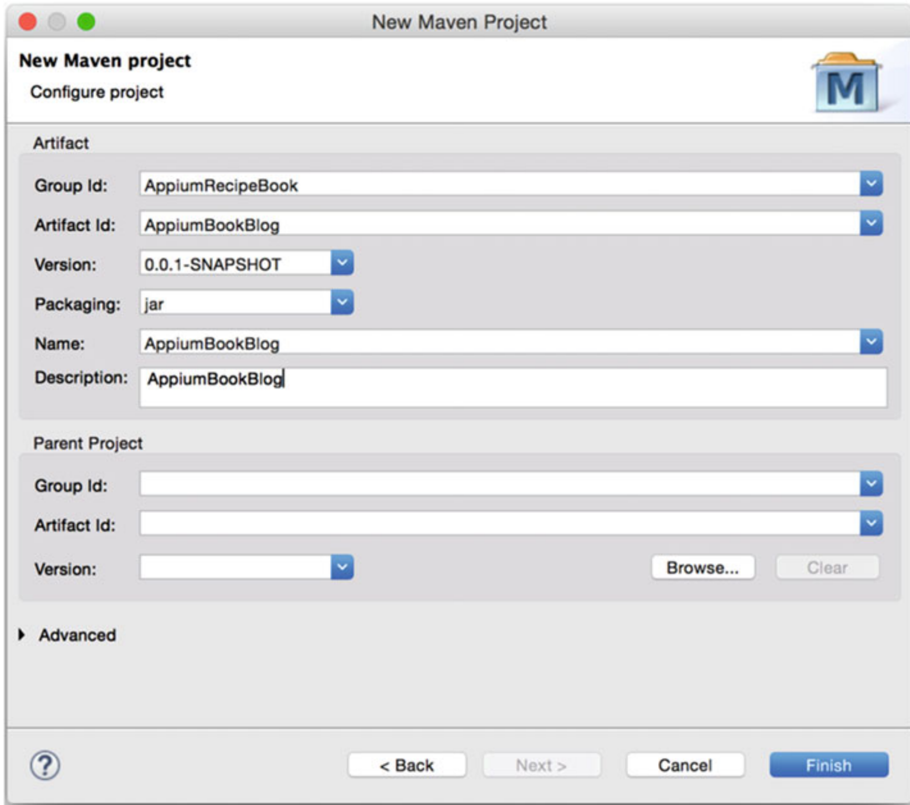


Figure 5-3. *Maven project creation wizard, project details*

6. This will create a basic Maven project. Update the `pom.xml` file with the following code to add Appium and TestNG dependencies:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>AppiumRecipeBook</groupId>
  <artifactId>AppiumBookBlog</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>AppiumBookBlog</name>
  <description>AppiumBookBlog</description>

   <properties>
            <appium.version>3.3.0</appium.version>
            <testng.version>6.9.10</testng.version>
            <selenium.version>2.47.1</selenium.
            version>
    </properties>
    <dependencies>
           <!--  Appium -->
           <dependency>
                  <groupId>io.appium</groupId>
                  <artifactId>java-client
                  </artifactId>
                  <version>${appium.version}
                  </version>
                  <scope>test</scope>
           </dependency>

           <!-- testng -->
           <dependency>
                  <groupId>org.testng</groupId>
                  <artifactId>testng</artifactId>
                  <version>${testng.version}
                  </version>
                  <scope>test</scope>
           </dependency>
    </dependencies>
</project>
```

7. To keep the code and files in logical grouping, you need to create some packages in the default project. Refer to Figure 5-4 and create the appium package in the src/test/java package and create the apps folder in the src/test/resources package (Figure 5-4).

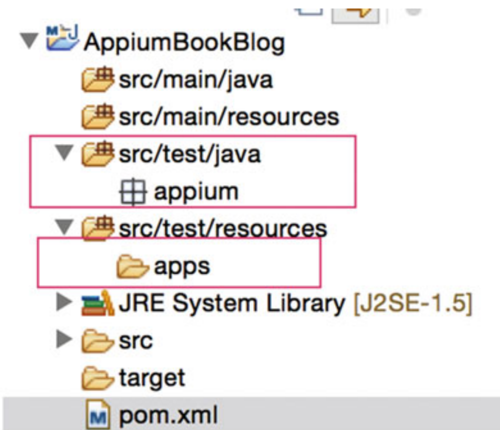


Figure 5-4. *Project structure of the sample project*

8. You will see an Android example for this project, so add the Android ApiDemos-debug.apk file to the apps folder.

9. Create a new class called AppiumDriverBase in the appium package. Add the following code to this class:

```
package appium;

import io.appium.java_client.android.AndroidDriver;

import java.io.File;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;

public class AppiumDriverBase {

    protected WebDriver driver;
    protected WebDriverWait wait;
```

```
    // before Test Annotation makes a java function to
    run every time before a TestNG test case
    @BeforeTest
    protected void createAppiumDriver() throws
    MalformedURLException, InterruptedException {

       // setting up desired capability
       DesiredCapabilities caps = new
       DesiredCapabilities();
       caps.setCapability("platform", "ANDROID");
       caps.setCapability("platformVersion", "5.0");
       caps.setCapability("deviceName", "ANDROID");
       caps.setCapability("browserName", "");

       // relative path to apk file
       final File classpathRoot = new File(System.
       getProperty("user.dir"));
       final File appDir = new File(classpathRoot,
       "src/test/resources/apps/");
       final File app = new File(appDir, "ApiDemos-
       debug.apk");
       caps.setCapability("app", app.
       getAbsolutePath());

       // initializing driver object
       driver = new AndroidDriver(new
       URL("http://127.0.0.1:4723/wd/hub"), caps);
       // initializing explicit wait object
       driver.manage().timeouts().implicitlyWait(10,
       TimeUnit.SECONDS);
       wait = new WebDriverWait(driver, 10);
    }
    // After Test Annotation makes a java function to
    run every time after a TestNG test case
    @AfterTest
    public void afterTest() {

       // quit the driver
       driver.quit();
    }

}
```

10. Add the test case class called `SampleTestCase` by creating one more class in the Appium package. Add the following code to this class:

```
package appium;

import io.appium.java_client.MobileBy;

import org.openqa.selenium.By;
import org.openqa.selenium.support.
ui.ExpectedConditions;
import org.testng.annotations.Test;

public class SampleTestCase extends AppiumDriverBase{

    //Test Annotation changes any java function to
    TestNG test case
    @Test
    public void sampeTest(){
       //click on Accessibility link
       wait.until(ExpectedConditions.presenceOfElement
       Located(MobileBy.AccessibilityId
       ("Accessibility")));
       driver.findElement(MobileBy.AccessibilityId
       ("Accessibility")).click();
       //click on 'Accessibility Node Querying' link
       wait.until(ExpectedConditions.presenceOfElement
       Located(MobileBy.AccessibilityId("Accessibility
       Node Querying")));
       driver.findElement(MobileBy.
       AccessibilityId("Accessibility Node Querying")).
       click();
       //back
       driver.navigate().back();
       //back
       driver.navigate().back();
    }
}
```

11. The first test case is ready. Run the Appium server on a terminal, run an Android emulator, and execute the program by right-clicking the file and selecting SampleTestCase ➤ Run As ➤ TestNG Test.

The Appium server should receive the request, and the program should be executed appropriately (Figure 5-5).

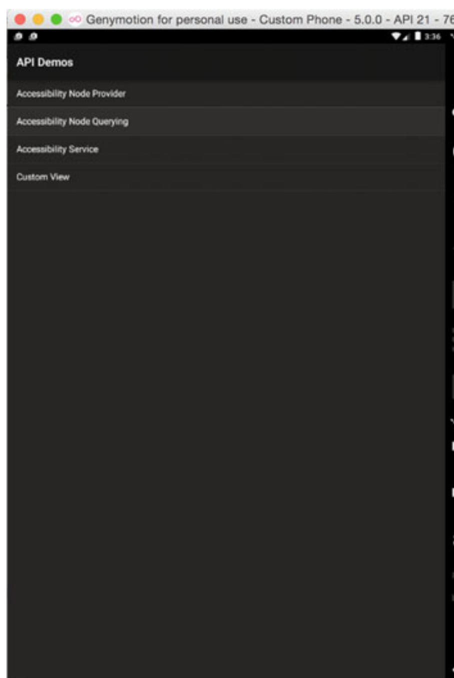


Figure 5-5. *Sample Android test case executed*

12. Now open a terminal and `cd` to the project root directory. Type the following command to execute all the methods tagged with the `@Test` annotation.

```
mvn test
```

The test case should execute successfully.

■ **Note** The only difference for iOS would be the initiation of the driver object in the `@BeforeTest` method; everything else remains the same for iOS and TestNG integration.

How It Works

Maven and TestNG are a popular combination for Appium. They make up the base of the test automation framework. The following are some reasons for integrating Maven and TestNG:

- Maven is a build tool that helps integrate all the required Java libraries mentioned in the `pom.xml` file as dependencies. The benefit is that in the case of any updates, you just need to update the dependency version, and Maven takes care of the rest.
- The Maven life cycle helps ease the execution part. Functions mentioned with the `@Test` tag in the test package can be easily executed with `mvn test` from a terminal.
- TestNG helps tag methods as test cases and also helps with the before and after methods. You just need to add as many methods as you need and tag them with an appropriate tag such as `@Test` or `@BeforeSuite`, `@BeforeTest`, and so on.
- TestNG creates a consolidated HTML report of the test results automatically without needing you to do anything. This complements one of the biggest shortcomings of Selenium-based tools.

■ **Note** Integrating Appium with JUnit is similar to integrating Appium with TestNG. The first difference is in `pom.xml`; you would add a dependency of JUnit instead of TestNG. That's all you need to do differently to start using the `@BeforeClass`, `@AfterClass`, and `@Test` tags of JUnit in Java.

5-2. Create a BDD Framework with Appium, Cucumber, and the Page Object Model

Problem

Behavior-driven development is gaining lot of popularity, and Cucumber is the best tool to implement BDD, so you want to understand how to integrate Cucumber and Appium.

The framework that you create should be easy to maintain and extend, so the industry best practice of the page object model should also be integrated in the framework. You want to learn how to do this.

Solution

You will create a behavior-driven development framework with Appium, Cucumber, and JUnit and write one sample test case for an iOS app.

1. Install the Eclipse-TestNG plug-in by following the steps at https://shankargarg.wordpress.com/2015/04/26/how-to-integrate-eclipse-with-cucumber-plugin/.

2. Create a simple Maven project using the Eclipse-Maven plug-in. Follow the steps until step 3 in recipe 5-1. Name the project AppiumCucumberPageObject.

3. For a simple Maven project, this is what the pom.xml file looks like:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>AppiumRecipeBook</groupId>
  <artifactId>AppiumCucumberPageObject</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>AppiumCucumberPageObject</name>
  <description>AppiumCucumberPageObject</description>

</project>
```

4. Now, you need to update the pom.xml file for the dependencies of Cucumber and Appium. First, add the properties tag and define properties for the Cucumber and Appium versions. This is done to make sure that when you need to update the dependency version, you do it in only one place in the properties.

```
<properties>
        <appium.version>4.0.0</appium.version>
        <cucumber.version>1.2.4</cucumber.version>
</properties>
```

■ **Note** Please use the Maven central repository at http://search.maven.org/ to check the latest dependency versions of Cucumber and Appium.

---

5. Add dependencies for `cucumber-java` and `cucumber-junit` for BDD and for `java-client` for mobile automation by using the following code:

```
<dependencies>
        <!-- cucumber -->
        <dependency>
            <groupId>info.cukes</groupId>
            <artifactId>cucumber-java</artifactId>
            <version>${cucumber.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>info.cukes</groupId>
            <artifactId>cucumber-junit</artifactId>
            <version>${cucumber.version}</version>
            <scope>test</scope>
        </dependency>

        <!--  Appium -->
        <dependency>
          <groupId>io.appium</groupId>
          <artifactId>java-client</artifactId>
          <version>${appium.version}</version>
        </dependency>
 </dependencies>
```

6. To keep the logical files in the same place, you will create some packages in the default project, such as the `stepdefinition` package to keep all the Cucumber step definitions and the `pages` package to keep all the page object files. Follow the setup in Figure 5-6 and create the packages as mentioned.

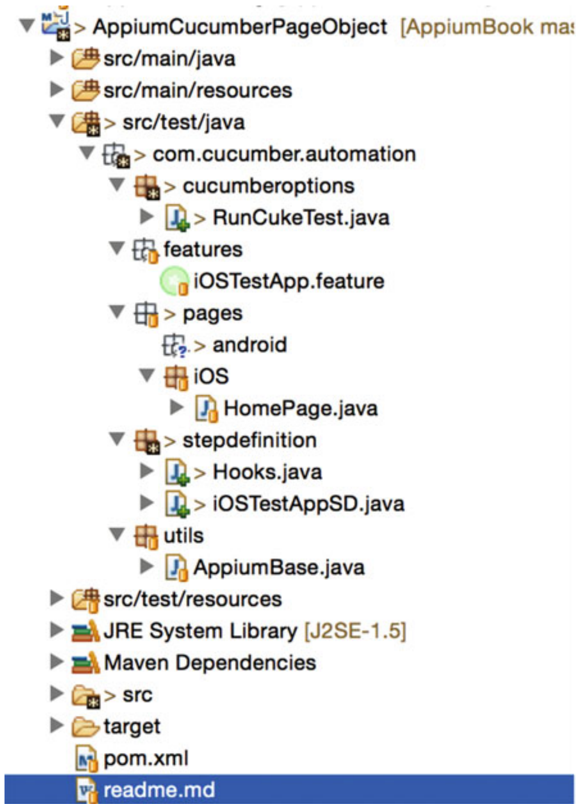


Figure 5-6. *Structure for AppiumCucumberPageObject project*

7. For a Cucumber project, the `RunCukeTest.java` file specifies the configuration such as the location of feature files, the location of step definitions, the output location, and so on. Add the `RunCukeTest` class to the `cucumberoptions` package with the following code:

```
package com.cucumber.automation.cucumberoptions;

import org.junit.runner.RunWith;

import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;

    @RunWith(Cucumber.class)
    @CucumberOptions(

              features = "src/test/java/com/cucumber/
              automation/features",
                  glue = "com.cucumber.automation.
                  stepdefinition",
              plugin = {
                                  "pretty",
                                  "html:target/cucumber",
                           }
             )
    public class RunCukeTest {
}
```

8. Requirements are set in feature files. Since you are using the iOS Test App for this demonstration, you will add the `iOSTestApp.feature` file to the package `features`. This is how the feature file will look:

```
Feature: iOS Test App
    In order to test sample ios app
    As a product owner
           I want to specify generic scenarios

Scenario: Calculate Sum
    Given user is on Application Home Page
    When user enters "4" in first field
    And user enters "5" in second field
    And clicks on Compute Sum
    Then user sees computed sum as "9"
```

9. The feature file has to be converted to StepDefinition for Cucumber to understand this file. The simplest way is to use the suggestions given by Cucumber. In `iOSTestApp.feature`, right-click and select Run As à Cucumber Feature. Now copy the suggestions given by Cucumber in the console output shown in Figure 5-7.

```
You can implement missing steps with the snippets below:

@Given("^user is on Application Home Page$")
public void user_is_on_Application_Home_Page() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^user enters \"([^\"]*)\" in first field$")
public void user_enters_in_first_field(String arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^user enters \"([^\"]*)\" in second field$")
public void user_enters_in_second_field(String arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^clicks on Compute Sum$")
public void clicks_on_Compute_Sum() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Then("^user sees computed sum as \"([^\"]*)\"$")
public void user_sees_computed_sum_as(String arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

Figure 5-7. *Cucumber's suggestion for step definitions*

10. Add a file called `iOSTestAppSD.java` to the `stepdefinition` package with the following code:

```
package com.cucumber.automation.stepdefinition;

import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;

public class iOSTestAppSD {

    @Given("^user is on Application Home Page$")
    public void user_is_on_Application_Home_Page() {
    }

    @When("^user enters \"([^\"]*)\" in first field$")
    public void user_enters_in_first_field(String arg1) {
    }

    @When("^user enters \"([^\"]*)\" in second field$")
    public void user_enters_in_second_field(String arg1) {
    }

    @When("^clicks on Compute Sum$")
    public void clicks_on_Compute_Sum() {
    }
    @Then("^user sees computed sum as \"([^\"]*)\"$")
    public void user_sees_computed_sum_as(String arg1) {
    }
}
```

11. You need to specify and add the test apps to be used for the test case execution. Add the `.apk/.app` files in the apps folder in the `src/test/resources` package.

12. Add the Appium functions that can be used to invoke the Android app and close the app once the execution finishes. (I am keeping this file basic for simplicity purposes.) Create the `AppiumBase.java` class in the `utils` package with the following code:

```
package com.cucumber.automation.utils;

import io.appium.java_client.ios.IOSDriver;

import java.io.File;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.concurrent.TimeUnit;
```

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.support.ui.WebDriverWait;

public class AppiumBase {

    public static WebDriver driver;
    public static WebDriverWait waitVar;

    public void createDriver() throws
    MalformedURLException, InterruptedException {
       // setting capabilities
       DesiredCapabilities caps = new
       DesiredCapabilities();
       caps.setCapability("platform", "iOS");
       caps.setCapability("platformVersion", "9.2");
       caps.setCapability("deviceName", "iPhone 6");

       // relative path to .app file
       final File classpathRoot = new File(System.
       getProperty("user.dir"));
       final File appDir = new File(classpathRoot,
       "src/test/resources/apps/");
       final File app = new File(appDir, "TestApp.
       app");
       caps.setCapability("app", app.
       getAbsolutePath());

       // initializing driver object
       driver = new IOSDriver(new
       URL("http://127.0.0.1:4723/wd/hub"), caps);

       // initializing waits
       driver.manage().timeouts().implicitlyWait(10,
       TimeUnit.SECONDS);
       waitVar = new WebDriverWait(driver, 10);
    }

    public void teardown() {
       // close the app
       driver.quit();
    }
}
```

13. You need to add a hooks file so that Cucumber can call functions placed in the AppiumBase file. Create the Hooks.java class in the stepdefinition package with the following code:

```
package com.cucumber.automation.stepdefinition;

import java.net.MalformedURLException;

import com.cucumber.automation.utils.AppiumBase;

import cucumber.api.java.After;
import cucumber.api.java.Before;

public class Hooks {

    AppiumBase appiumBase = new AppiumBase();

    @Before
    public void beforeHookfunction() throws
    MalformedURLException, InterruptedException{
        appiumBase.createDriver();
    }

    @After
    public void afterHookfunction() {
        appiumBase.teardown();
    }

}
```

14. Let's start implementing the page object model (POM). I am keeping the POM simple, but you are free to extend it as per your requirements. For this iOS app, since there is only one screen, you will add only one page called HomePage.java in the pages.iOS package with the following code:

```
package com.cucumber.automation.pages.iOS;

import io.appium.java_client.MobileBy;

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

import com.cucumber.automation.utils.AppiumBase;

public class HomePage extends AppiumBase{

    // All the locators for Home page will be defined
    here
    By textField1 = MobileBy.AccessibilityId("TextFie
    ld1");
```

```
    By textField2 = MobileBy.AccessibilityId("TextFie
    ld2");
    By computeSum = MobileBy.AccessibilityId("Compute
    SumButton");
    By result = MobileBy.AccessibilityId("Answer");

    // All the behavior of home page will be defined
    here in functions
    public boolean isHomePage(){
        waitVar.until(ExpectedConditions.presenceOfEle
        mentLocated(computeSum));
        return driver.findElement(computeSum).
        isDisplayed();
    }

    public void typeTextField1(String text){
         waitVar.until(ExpectedConditions.presenceOfEle
         mentLocated(textField1));
         driver.findElement(textField1).sendKeys(text);
    }

    public void typeTextField2(String text){
         waitVar.until(ExpectedConditions.presenceOfEle
         mentLocated(textField2));
         driver.findElement(textField2).sendKeys(text);
    }

    public void clickComputeSum(){
         waitVar.until(ExpectedConditions.presenceOfEle
         mentLocated(computeSum));
         driver.findElement(computeSum).click();
    }

    public String returnResult(){
         waitVar.until(ExpectedConditions.presenceOfEle
         mentLocated(result));
         return driver.findElement(result).getText();
    }
}
```

15. You will have to update the step definition files for the Appium functions that you have just written. After adding all the functions, the code should look like this:

```
package com.cucumber.automation.stepdefinition;

import com.cucumber.automation.pages.iOS.HomePage;

import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;

import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertEquals;

public class iOSTestAppSD {

    HomePage homePage = new HomePage();

    @Given("^user is on Application Home Page$")
    public void user_is_on_Application_Home_Page() {
        assertTrue(homePage.isHomePage());
    }

    @When("^user enters \"([^\"]*)\" in first field$")
    public void user_enters_in_first_field(String arg1)
{
        homePage.typeTextField1(arg1);
    }

    @When("^user enters \"([^\"]*)\" in second field$")
    public void user_enters_in_second_field(String
    arg1) {
        homePage.typeTextField2(arg1);
    }

    @When("^clicks on Compute Sum$")
    public void clicks_on_Compute_Sum() {
        homePage.clickComputeSum();
    }

    @Then("^user sees computed sum as \"([^\"]*)\"$")
    public void user_sees_computed_sum_as(String arg1)
{
        assertEquals(arg1, homePage.returnResult());
    }

}
```

16. The framework is ready. Run the Appium server on a terminal and execute the program by going to `iOSTestApp.feature`. Then right-click Run As and select Cucumber Feature.

    The Appium server should receive the request, and the program should be executed appropriately (Figure 5-8).

```
Feature: iOS Test App
    In order to test sample ios app
    As a product owner
    I want to specify generic scenarios
1
2

  Scenario: Calculate Sum                  # /Users/sgarg/Documents/xebia/Docs/appium/AppiumCookBook/code/Ap
    Given user is on Application Home Page # iOSTestAppSD.user_is_on_Application_Home_Page()
    When user enters "4" in first field    # iOSTestAppSD.user_enters_in_first_field(String)
    And user enters "5" in second field    # iOSTestAppSD.user_enters_in_second_field(String)
    And clicks on Compute Sum              # iOSTestAppSD.clicks_on_Compute_Sum()
    Then user sees computed sum as "9"     # iOSTestAppSD.user_sees_computed_sum_as(String)

1 Scenarios (1 passed)
5 Steps (5 passed)
1m10.940s
```

Figure 5-8. *Console output for the Appium Cucumber project*

17. You can open a terminal and `cd` to the project root directory, typing the following command to execute all the scenarios in all the feature files:

    ```
    mvn test
    ```

    The scenarios should execute fine.

■ **Note** The only difference for Android would be to initiate the driver object in the `@Before` method; everything else remains the same for integrating Android with Cucumber.

How It Works

You have integrated Cucumber, Appium, Maven, Java, and page objects to design your mobile automation frameworks. Cucumber is for implementing BDD so that nontechnical people can also directly contribute to development, Appium is for web automation, Java is a programming language, and Maven is a build tool.

The page object model is a framework design approach for maintaining and accessing components and controls spread across test scenarios. The page object model creates a domain-specific language (DSL) for your tests so that if something changes on the page, you don't need to change the test; you just need to update the object that represents the page.

5-3. Conduct Continuous Automated Testing with Appium, Git, and Jenkins

Problem

A test automation framework should integrate with version control and continuous integration tools so that the latest test code can always be executed either on-demand or at a scheduled time. You want to know how to integrate the Appium framework with a version control management system and a continuous integration tool.

Solution

Git is the most famous version control management system. GitHub is most popular version of it and is available for free for a certain number of users. So, for version control in this recipe, you are going to use GitHub.

Jenkins is most popular tool available for automated build and continuous integration. Jenkins has lot of advantages as it is open source, free, and easy to use, and it can schedule a run at a scheduled time or trigger builds after an event.

Installing Jenkins and GitHub is beyond the scope of this book, and thus I am assuming that you have Jenkins and Git already installed and set up.

- If you need any help regarding Jenkins setup, please follow the steps here: `https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins`
- You need to upload the project on GitHub. If you need any help in Git or with the GitHub repo setup, then follow these instructions: `https://help.github.com/articles/set-up-git/`
- You can download the projects that will be used in this recipe from the following GitHub URL: `https://github.com/ShankarGarg/AppiumBook.git`

To get started, you will run Jenkins locally and execute the `AppiumBookBlog` project created in recipe 5-1 by taking the latest code from the GitHub repository.

1. Use the http://0.0.0.0:8080/ URL to open Jenkins in any browser (replace `0.0.0.0` or `localhost` with the machine IP address if Jenkins is not running locally).

2. Go to the Jenkins dashboard and click New Item (Figure 5-9).

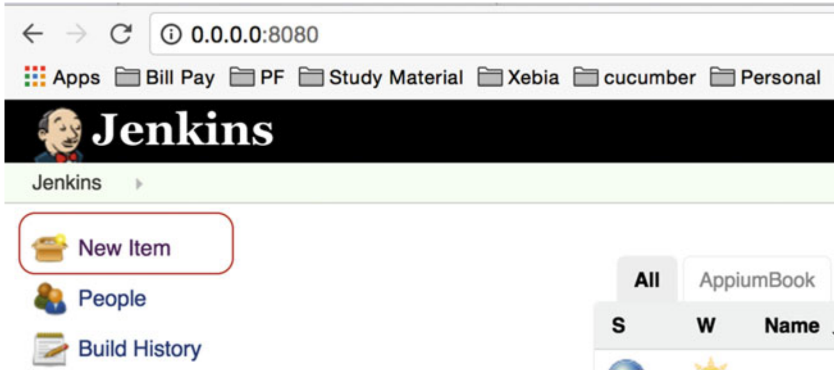


Figure 5-9. *New Item on Jenkins dashboard*

3. Enter the Jenkins job name that you want to create, select the "Maven project" option, and click OK (Figure 5-10).

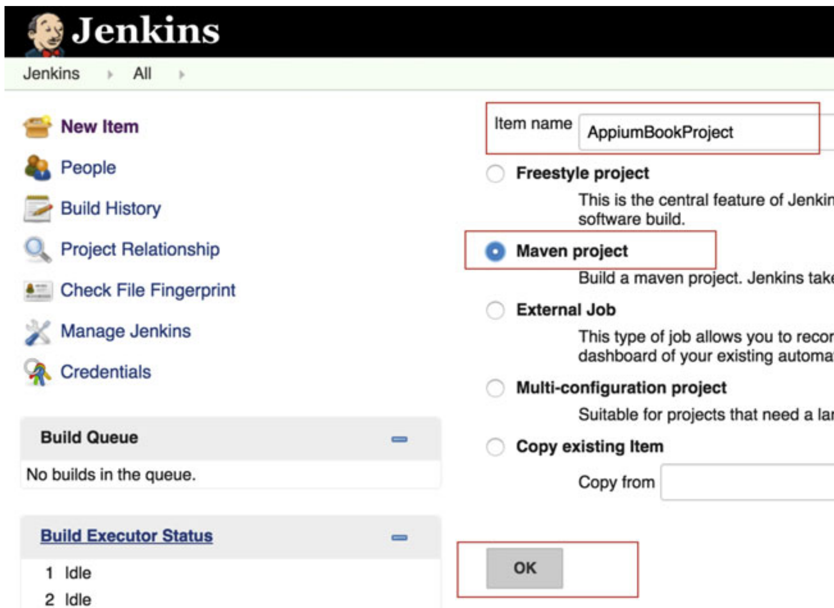


Figure 5-10. *Project name for the Jenkins job*

4. Enter a description of the project (Figure 5-11).

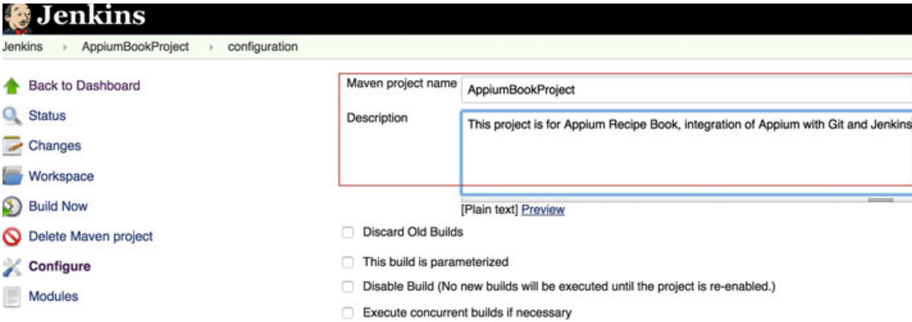


Figure 5-11. *Project description*

5. In Source Code Management section, select Git, fill in the Repository URL field as **`https://github.com/ShankarGarg/AppiumBook.git`**, and fill in your GitHub credentials (Figure 5-12). Keep the others options in this section set to their defaults.

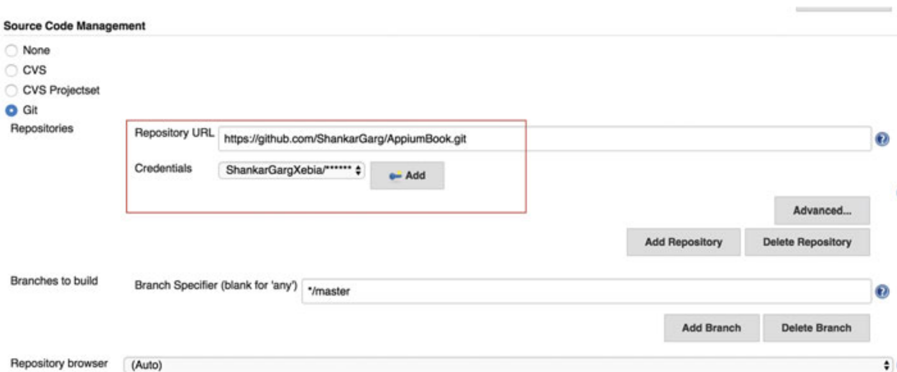


Figure 5-12. *GitHub repository and credentials*

6. In the Build section, since this is a Maven project, the root `pom.xml` file is automatically mentioned, but since the repository has multiple projects, you need to refer to the exact `pom.xml` file that you want to run in this project. Also, you need to mention the goal `test` that you want to run in this project (Figure 5-13).

Build

Root POM	AppiumBookBlog/pom.xml
Goals and options	test

Advanced...

Figure 5-13. *pom.xml and Maven goal*

7. Keep all other options set to their defaults and click Save. You will be redirected to the dashboard of the newly created Jenkins project.

8. On this page, click Build Now to run the project (Figure 5-14).

Jenkins

Jenkins › AppiumBookProject ›

Back to Dashboard
Status
Changes
Workspace
Build Now
Delete Maven project
Configure
Modules

Build History trend

RSS for all RSS for failures

Maven project AppiumBookProject

This project is for Appium Recipe Book, integration of Appium with Git and Jenkins

Workspace

Recent Changes

Permalinks

Figure 5-14. *Building the project*

9. Once you click Build Now, a build is triggered immediately. You can see the build number and the timestamp (Figure 5-15).

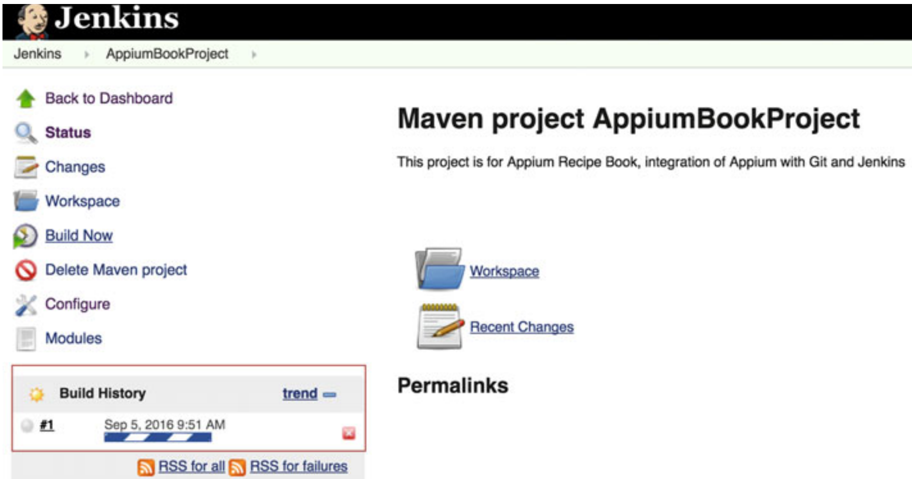


Figure 5-15. *Build number and timestamp*

10. Click the timestamp on the build. And then click Console Output to see the console output of the project (Figure 5-16).

Figure 5-16. *Console output for the Appium project*

How It Works

You have already integrated Appium with Maven, so integrating Appium with Jenkins just meant running the Appium Maven project via Jenkins. Jenkins comes with a Maven plug-in by default; when you selected the item type of building a Maven project, most of the settings were taken care of then. The Build section was prepopulated with `pom.xml`, and you just had to select the appropriate `pom.xml` file and set the goal to `test`.

Jenkins is also prepopulated with the GitHub plug-in, so you just had to set the GitHub URL and credentials. Now every time the project is built, Jenkins takes the latest code from Git and then runs the test cases.

■ **Note** To explore more, you can go to the Build Triggers section in the Jenkins job and try to schedule the job with various configurations.

CHAPTER 6

Integrating Appium with Selenium Grid

In this chapter, you will learn to Integrate:

- Appium with Selenium Grid for native app automation
- Appium with Selenium Grid for mobile web automation
- Appium with Selenium Grid for two Android sessions on the same machine

In previous chapters, you learned to create a test automation framework using Appium and integrate it with GitHub and Jenkins to schedule the test execution at desired times.

The last piece of the puzzle for an effective test strategy is to optimize the test infrastructure. Either you can execute mobile test cases on simulators and real devices on the local infrastructure managed by your company/client or you can use simulators and real devices on cloud test labs provided by vendors such as Sauce Labs and Testdroid.

The decision to use the local infrastructure versus a cloud lab depends on a lot of factors such as cost and the effort required to set up and maintain a mobile test infrastructure. This decision is beyond the scope of this book. In this chapter, you will learn what it takes technically to run Appium test cases on the local infrastructure.

6-1. Appium with Selenium Grid for Native App Automation

Problem

You have most of your test cases ready, so you want to execute them on multiple devices, and you want to create a test infrastructure that redirects the test cases to the appropriate device based on the desired capabilities in the test case.

© Shankar Garg 2016
S. Garg, *Appium Recipes*, DOI 10.1007/978-1-4842-2418-2_6

Solution

In this recipe, you will set up Selenium Grid to redirect the test cases to the appropriate device based on the desired capabilities in the test case. The scope of this recipe is for native app automation for both the Android and iOS platforms.

1. Download the Selenium server JAR from `http://selenium-release.storage.googleapis.com/index.html`. I have used version 2.53.1 for this book. It has been saved in the `src/test/resources/drivers` folder of the `AppiumRecipeBook` project.

2. Open a terminal, `cd` to the `AppiumRecipesBook/src/test/resources/drivers` folder, and run the following command to start the Selenium server:

```
java -Djava.net.preferIPv4Stack=false -jar selenium-
server-standalone-2.53.1.jar -role hub
```

■ **Note** 2.53.1 is a stable version, but the version will vary as per updates in the Selenium release.

`-Djava.net.preferIPv4Stack=false` is to set my machine's Java to accept connections properly. Try using the previous command without this property, and if it works fine for you, then there's no need to use it.

---

3. The Selenium Grid terminal output should look like Figure 6-1. Open `http://192.168.56.1:4444/grid/console` in a browser to check the grid configurations and nodes (Figure 6-2).

```
Shankars-MacBook-Pro:drivers sgarg$ java -Djava.net.preferIPv4Stack=false -jar selenium-server-standalone-2.53.1.jar -rol
e hub
12:19:55.008 INFO - Launching Selenium Grid hub
2016-09-11 12:19:55.722:INFO::main: Logging initialized @866ms
12:19:55.733 INFO - Will listen on 4444
12:19:55.781 INFO - Will listen on 4444
2016-09-11 12:19:55.783:INFO:osjs.Server:main: jetty-9.2.z-SNAPSHOT
2016-09-11 12:19:55.812:INFO:osjsh.ContextHandler:main: Started o.s.j.s.ServletContextHandler@28ac3dc3{/,null,AVAILABLE}
2016-09-11 12:19:55.842:INFO:osjs.ServerConnector:main: Started ServerConnector@25bbf683{HTTP/1.1}{0.0.0.0:4444}
2016-09-11 12:19:55.842:INFO:osjs.Server:main: Started @987ms
12:19:55.843 INFO - Nodes should register to http://192.168.56.1:4444/grid/register/
12:19:55.843 INFO - Selenium Grid hub is up and running
```

Figure 6-1. *Terminal output for Selenium Grid*

Figure 6-2. *Grid console for Selenium Grid*

4. Now Selenium Grid is ready to listen to requests at `http://192.168.56.1:4444/wd/hub` from Appium instances.

■ **Note** 192.168.56.1 is the IP address of my machine; you can also use `localhost` for simplicity. If Selenium Grid is running remotely on another machine, then you need to use the IP address of that machine.

Android

Now you need to create an Appium instance that will act as a slave/node to the Selenium server setup in the previous steps. You will create a node configuration file called `AppiumNodeConfigAndroidNative.json` that will contain all the properties that this node session will have.

1. Create a file called `AppiumNodeConfigAndroidNative.json` in the `src/test/resources/AppiumConfig` package in the `AppiumRecipesBook` project with the following content to create an Android native instance:

```
{
"capabilities":
      [
        {
              "maxInstances": 1,
              "browserName": "",
              "platform":"android",
              "version":"5.1"
        }
      ],
  "configuration":
  {
    "cleanUpCycle":2000,
```

```
        "timeout":30000,
        "proxy": "org.openqa.grid.selenium.proxy.
         DefaultRemoteProxy",
        "hub": "http://192.168.56.1:4444/grid/register",
        "url":"http://127.0.0.1:4723/wd/hub",
        "host": "127.0.0.1",
        "port": 4723,
        "maxSession": 1,
        "register": true,
        "registerCycle": 5000,
        "hubPort": 4444,
        "hubHost": "192.168.56.1",
        "role": "node",
        "throwOnCapabilityNotPresent":"false"
      }
    }
```

2. To start the Appium node session, open a new terminal, `cd` to the `/src/test/resources/AppiumConfig` folder in the `AppiumRecipesBook` project, and start Appium with the following command:

   ```
   appium --nodeconfig AppiumNodeConfigAndroidNative.json
   ```

   The console output of the previous command should look like Figure 6-3.

```
Shankars-MacBook-Pro:AppiumConfig sgarg$ appium --nodeconfig AppiumNodeConfigAndroidNative.json
[Appium] Welcome to Appium v1.5.0 (REV e6f1500728e48f4be59bb4ca2b476198f6559840)
[Appium] Non-default server args:
[Appium]   nodeconfig: 'AppiumNodeConfigAndroidNative.json'
[debug] [Appium] Starting auto register thread for grid. Will try to register every 5000 ms.
[Appium] Appium REST http interface listener started on 0.0.0.0:4723
[debug] [Appium] Appium successfully registered with the grid on 192.168.56.1:4444
[HTTP] --> GET /wd/hub/status
[MJSONWP] Calling AppiumDriver.getStatus() with args: []
[MJSONWP] Responding to client with driver.getStatus() result: {"build":{"version":"1.5.0","revision":"e6f1500728e48f4be59bb4ca2b476198f
6559840"}}
[HTTP] <-- GET /wd/hub/status 200 15 ms - 121
[HTTP] --> GET /wd/hub/status
[MJSONWP] Calling AppiumDriver.getStatus() with args: []
[MJSONWP] Responding to client with driver.getStatus() result: {"build":{"version":"1.5.0","revision":"e6f1500728e48f4be59bb4ca2b476198f
6559840"}}
[HTTP] <-- GET /wd/hub/status 200 9 ms - 121
[HTTP] --> GET /wd/hub/status
```

Figure 6-3. *Console output for Appium Android native node registration*

3. The Selenium Grid terminal output should look like Figure 6-4, and the Selenium console at `http://192.168.56.1:4444/grid/console`# should show the newly registered node with its configurations (Figure 6-5).

```
Shankars-MacBook-Pro:drivers sgarg$ java -Djava.net.preferIPv4Stack=false -jar selenium-server-standalone-2.53.1.jar -rol
e hub
12:19:55.008 INFO - Launching Selenium Grid hub
2016-09-11 12:19:55.722:INFO::main: Logging initialized @866ms
12:19:55.733 INFO - Will listen on 4444
12:19:55.781 INFO - Will listen on 4444
2016-09-11 12:19:55.783:INFO:osjs.Server:main: jetty-9.2.z-SNAPSHOT
2016-09-11 12:19:55.812:INFO:osjsh.ContextHandler:main: Started o.s.j.s.ServletContextHandler@28ac3dc3{/,null,AVAILABLE}
2016-09-11 12:19:55.842:INFO:osjs.ServerConnector:main: Started ServerConnector@25bbf683{HTTP/1.1}{0.0.0.0:4444}
2016-09-11 12:19:55.842:INFO:osjs.Server:main: Started @987ms
12:19:55.843 INFO - Nodes should register to http://192.168.56.1:4444/grid/register/
12:19:55.843 INFO - Selenium Grid hub is up and running
12:45:04.586 INFO - Registered a node http://127.0.0.1:4723
```

Figure 6-4. *Selenium Grid terminal output for Selenium Grid registering a new node*

192.168.56.1:4444/grid/console#

Grid Console v.2.53.1

DefaultRemoteProxy (version : 1.5.0)
id : http://127.0.0.1:4723, OS : ANDROID

Browsers | Configuration

```
role:node
remoteHost:http://127.0.0.1:4723
hubHost:192.168.56.1
hubPort:4444
prioritizer:null
timeout:30000
throwOnCapabilityNotPresent:false
nodePolling:5000
url:http://127.0.0.1:4723/wd/hub
newSessionWaitTimeout:-1
proxy:org.openqa.grid.selenium.proxy.DefaultRemoteProxy
cleanUpCycle:2000
hub:http://192.168.56.1:4444/grid/register
port:4723
browserTimeout:0
host:127.0.0.1
servlets:[]
maxSession:1
registerCycle:5000
capabilityMatcher:org.openqa.grid.internal.utils.DefaultCapabilityMatcher
jettyMaxThreads:-1
register:true
```

view config

Figure 6-5. *Selenium Grid console: node details*

The server and node are ready, and now you need to make changes in your test case to redirect the test cases to Selenium Grid, instead of just going to the Appium server.

4. You will use your test case created in earlier chapters, `AppiumSampleTestCaseAndroid`, and make the necessary changes to execute it using the Selenium Grid setup. Replace the line where you create the driver object with the following suggestion:

```
//Line to be replaced:
driver = new AndroidDriver(new
URL("http://127.0.0.1:4723/wd/hub"), caps);
```

```
//New Line to be added - Driver object with Grid
address
driver = new AndroidDriver(new
URL("http://192.168.56.1:4444/wd/hub"), caps);
```

■ **Note** To dynamically switch between local and Selenium Grid execution, you can pass a command-line argument to specify executing on local or on Selenium Grid.

---

5. Execute the program as explained in the previous chapters. Selenium Grid should receive the request, create a new session, and redirect the request appropriately (Figure 6-6).

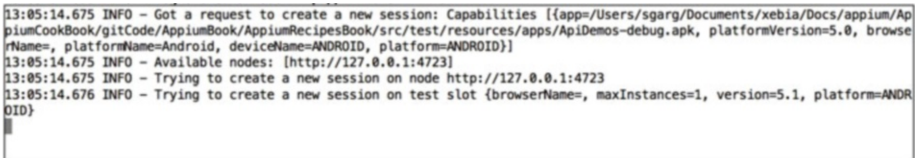


Figure 6-6. *Selenium Grid response to a new session for Android native apps*

The Appium node should receive the request (Figure 6-7), and the program should be executed appropriately (Figure 6-8).

```
[Appium] Creating new AndroidDriver session
[Appium] Capabilities:
[Appium]   app: '/Users/sgarg/Documents/xebia/Docs/appium/AppiumCookBook/gitCode/AppiumBook/AppiumRecipesBook/src/test/resources/apps/Ap
iDemos-debug.apk'
[Appium]   platformVersion: '5.0'
[Appium]   browserName: ''
[Appium]   platformName: 'Android'
[Appium]   deviceName: 'ANDROID'
[Appium]   platform: 'ANDROID'
[BaseDriver] The following capabilities were provided, but are not recognized by appium: platform.
[BaseDriver] Session created with session id: 5ecb4b66-5867-4405-9041-e13aafddf584
[debug] [AndroidDriver] Getting Java version
[AndroidDriver] Java version is: 1.8.0_51
[ADB] Checking whether adb is present
[ADB] Using adb from /Users/sgarg/Documents/Softwares/android/platform-tools/adb
[AndroidDriver] Retrieving device list
[debug] [ADB] Trying to find a connected android device
[debug] [ADB] Getting connected devices...
[debug] [ADB] 0 device(s) connected
[debug] [ADB] Could not find devices, restarting adb server...
[debug] [ADB] Restarting adb
[debug] [ADB] Getting connected devices...
[HTTP] --> GET /wd/hub/status
[MJSONWP] Calling AppiumDriver.getStatus() with args: []
[debug] [ADB] 1 device(s) connected
```

***Figure 6-7.** Appium node console output for a new session*

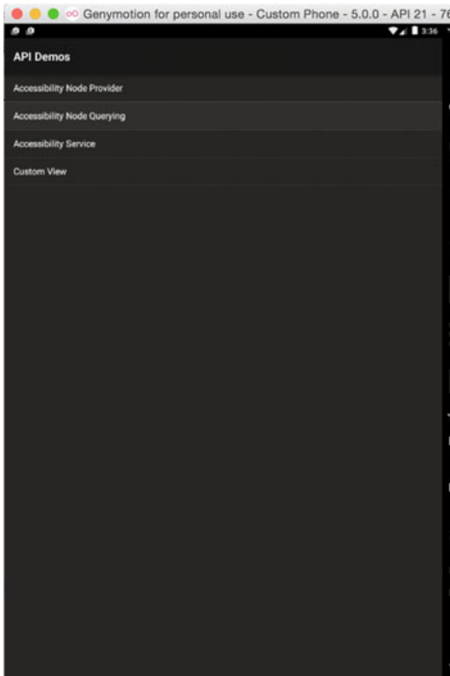


***Figure 6-8.** Android test case execution on an Android emulator*

iOS

Now you need to create an Appium instance that will act as an iOS node to Selenium Grid. You will create a node configuration file called `AppiumNodeConfigIOSNative.json` that will contain all the properties for this session.

You will use same grid setup as explained in steps 1–3 of the previous steps.

1. Create a file called `AppiumNodeConfigIOSNative.json` in the `src/test/resources/AppiumConfig` package in the `AppiumRecipesBook` project with the following content to create an Android iOS instance:

```
{
"capabilities":
      [
        {
        "maxInstances": 1,
         "browserName": "",
        "version": "9.2",
        "platformName": "iOS",
        "app": "/Users/sgarg/
        Documents/xebia/Docs/appium/AppiumCookBook/
        gitCode/AppiumBook/AppiumRecipesBook/src/test/
        resources/apps/TestApp.app",
        "newCommandTimeout":999
        }
      ],
  "configuration":
  {
    "cleanUpCycle":2000,
    "timeout":30000,
    "proxy": "org.openqa.grid.selenium.proxy.
    DefaultRemoteProxy",
    "hub": "http://192.168.56.1:4444/grid/register",
    "url":"http://127.0.0.1:4723/wd/hub",
    "host": "127.0.0.1",
    "port": 4723,
    "maxSession": 1,
    "register": true,
    "registerCycle": 5000,
    "hubPort": 4444,
    "hubHost": "192.168.56.1",
    "role": "node",
    "throwOnCapabilityNotPresent":"false"
  }
}
```

■ **Note** In app here, I am specifying the absolute path of the application under test; you should specify the path of your respective local folder structure.

---

2. To start the Appium node session, open a new terminal, cd to the /src/test/resources/AppiumConfig folder in the AppiumRecipesBook project, and start Appium with the following command:

```
appium --nodeconfig AppiumNodeConfigIOSNative.json
```

   The console output of the previous command should look like Figure 6-9.

```
Shankars-MacBook-Pro:AppiumConfig sgarg$ appium --nodeconfig AppiumNodeConfigIOSNative.json
[Appium] Welcome to Appium v1.5.0 (REV e6f1500728e48f4be59bb4ca2b476198f6559840)
[Appium] Non-default server args:
[Appium]   nodeconfig: 'AppiumNodeConfigIOSNative.json'
[debug] [Appium] Starting auto register thread for grid. Will try to register every 5000 ms.
[Appium] Appium REST http interface listener started on 0.0.0.0:4723
[debug] [Appium] Appium successfully registered with the grid on 192.168.56.1:4444
[HTTP] --> GET /wd/hub/status
[MJSONWP] Calling AppiumDriver.getStatus() with args: []
[MJSONWP] Responding to client with driver.getStatus() result: {"build":{"version":"1.5.0","revision":"e6f1500728e48f4be59bb4
ca2b476198f6559840"}}
[HTTP] <-- GET /wd/hub/status 200 16 ms - 121
```

Figure 6-9. Console output for Appium node registration

3. The Selenium Grid console output should look like Figure 6-4, and the Selenium console at http://192.168.56.1:4444/grid/console# should show the newly registered node with its configurations (Figure 6-10).

```
13:55:33.091 WARN - Cleaning up stale test sessions on the unregistered node http://127.0.0.1:4723
13:56:57.538 INFO - Registered a node http://127.0.0.1:4723
```

Figure 6-10. Selenium Grid terminal output for Selenium Grid registering a new node

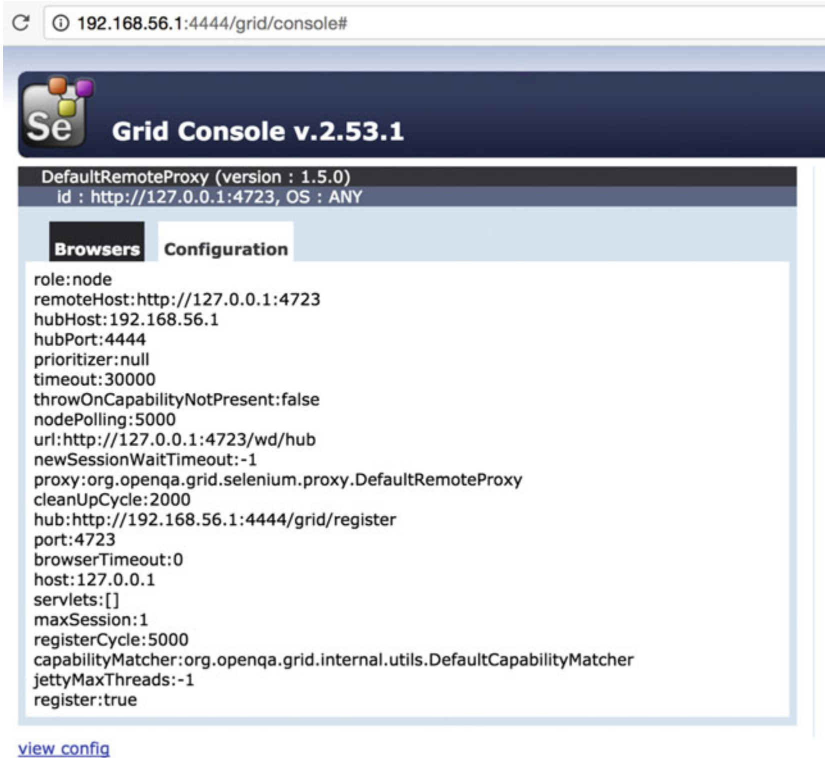


Figure 6-11. *Selenium Grid console: node details*

The server and node are ready, and now you need to make changes in your test case to redirect the test cases to Selenium Grid, instead of just going to the Appium server.

4. You will use the test case created in earlier chapters, `AppiumSampleTestCaseIOS`, and make the necessary changes to execute the test case using the Grid Selenium setup. Replace the line where you create the driver object with this suggestion:

```
//Line to be replaced:
driver = new IOSDriver(new URL("http://127.0.0.1:4723/wd/hub"), caps);

//New Line to be added - Driver object with Grid address
driver =  new IOSDriver(new URL("http://192.168.56.1:4444/wd/hub"), caps);
```

5. Execute the program as explained in the previous chapters. Selenium Grid should receive the request, create a new session, and redirect the request appropriately (Figure 6-12).

```
13:56:57.538 INFO - Registered a node http://127.0.0.1:4723
14:02:37.187 INFO - Got a request to create a new session: Capabilities [{app=/Users/sgarg/Documents/xebia/Docs/appium/Ap
piumCookBook/gitCode/AppiumBook/AppiumRecipesBook/src/test/resources/apps/TestApp.app, platformVersion=9.2, platformName=
iOS, deviceName=iPhone 6, platform=iOS}]
```

Figure 6-12. *Selenium Grid response to a new session*

The Appium node should receive the request, and the program should be executed appropriately (Figure 6-13).

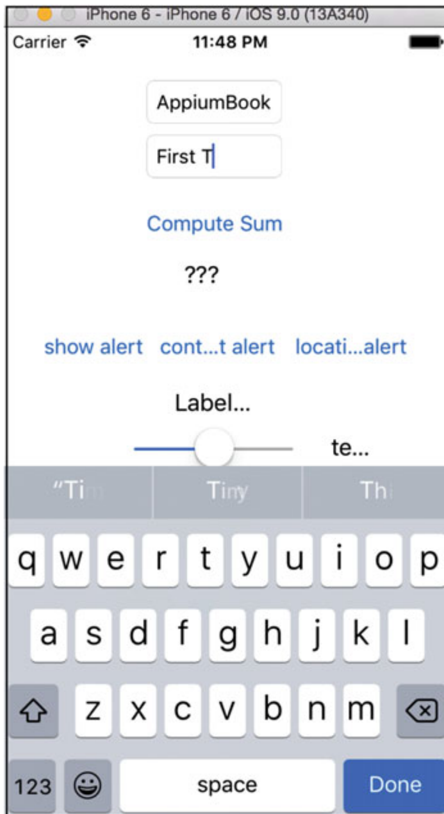


Figure 6-13. *iOS test case execution*

How It Works

When a test case is executed, Selenium Grid receives a request with certain desired capabilities, and then it redirects that request to an Appium instance/node session with the matching capabilities. So, in this recipe, those sessions would be either for Android native apps or for iOS native apps. Figure 6-14 shows the Appium grid architecture.

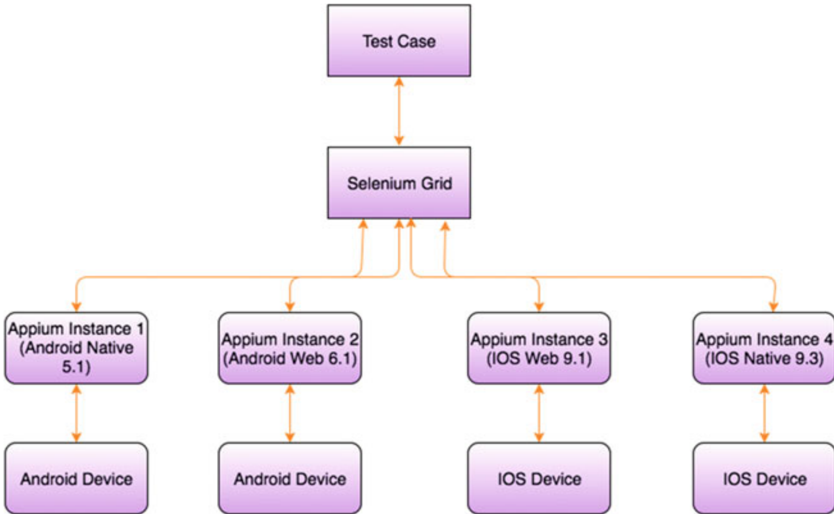


Figure 6-14. *Appium grid architecture*

Selenium Grid is for managing the redirects to the appropriate device based on the desired capabilities in the test case. You need to know the `hubPort` and `hubHost` settings to connect with Grid. In this case, these details are as follows:

- `hubPort`: 4444
- `hubHost`: 192.168.56.1

Appium instances/servers are for acting as the node, which receives the request from Selenium Grid, and then for interacting with the device for the test case execution. These communications are in the JSON wire protocol.

The file `config.json` is used to specify all the properties of the instance/session. One part of this file is to match the desired capabilities for an Appium session as follows:

```
"capabilities":
      [
        {
              "browserName": "",
              "platform":"android",
              "version":"5.1"
        }
      ]
```

Here, all the rules and knowledge for the desired capabilities will be applicable. The other part of the config file is the configuration, as shown here:

```
"configuration":
  {
    "cleanUpCycle":2000,
    "timeout":30000,
    "proxy": "org.openqa.grid.selenium.proxy.DefaultRemoteProxy",
    "hub": "http://192.168.56.1:4444/grid/register",
    "url":"http://127.0.0.1:4723/wd/hub",
    "host": "127.0.0.1",
    "port": 4723,
    "maxSession": 1,
    "register": true,
    "registerCycle": 5000,
    "hubPort": 4444,
    "hubHost": "192.168.56.1",
    "role": "node",
    "throwOnCapabilityNotPresent":"false"
  }
```

This specifies important configurations such as `role` as `node`, `hubHost` and `hubPort` for Selenium Grid, URL for the Appium server, `timeout`, cleanup, `registercycle` time limits, and so on.

When you start the Appium session, you need to specify that the current Appium session will use the properties from the config JSON file instead of the default values. You use the `-nodeconfig` parameter to provide the absolute path to the `config.json` file.

You need to redirect the test case to Selenium Grid, and that's why you change the URL of the driver object to the Selenium Grid URL: `http://192.168.56.1:4444/wd/hub`.

6-2. Appium with Selenium Grid for Mobile Web Automation

Problem

In the previous recipe, you learned to set up native app sessions with Selenium Grid. Now you want to set up mobile web sessions with Selenium Grid.

Solution

In this recipe, you will set up Selenium Grid for mobile web sessions on Android and iOS (in other words, Chrome on Android and Safari on iOS). For the Android and iOS solutions presented here, you will use same grid setup as steps 1–3 in recipe 6-1.

Android

You need to create an Appium instance for Chrome on Android that will act as a slave/node to Selenium Grid. You will create a node configuration file called `AppiumNodeConfigAndroidWeb.json` that will contain all the properties for this session.

1. Create a file called `AppiumNodeConfigAndroidWeb.json` in the `src/test/resources/AppiumConfig` package in the `AppiumRecipesBook` project with the following content:

```
{
"capabilities":
      [
        {
               "maxInstances": 1,
              "browserName": "chrome",
              "platform":"android",
              "version":"5.1"
        }
      ],
  "configuration":
  {
    "cleanUpCycle":2000,
    "timeout":30000,
    "proxy": "org.openqa.grid.selenium.proxy.
     DefaultRemoteProxy",
    "hub": "http://192.168.56.1:4444/grid/register",
    "url":"http://127.0.0.1:4723/wd/hub",
    "host": "127.0.0.1",
    "port": 4723,
    "maxSession": 1,
    "register": true,
    "registerCycle": 5000,
    "hubPort": 4444,
    "hubHost": "192.168.56.1",
    "role": "node",
    "throwOnCapabilityNotPresent":"false"
  }
}
```

2. To start an Appium node session, open a new terminal, `cd` to the `/src/test/resources/AppiumConfig` folder in the `AppiumRecipesBook` project, and start Appium with the following command:

```
appium --nodeconfig AppiumNodeConfigAndroidWeb.json
```

The console output of the previous command should look like Figure 6-15.

```
Shankars-MacBook-Pro:AppiumConfig sgarg$ appium  --nodeconfig AppiumNodeConfigAndroidWeb.json
[Appium] Welcome to Appium v1.5.0 (REV e6f1500728e48f4be59bb4ca2b476198f6559840)
[Appium] Non-default server args:
[Appium]   nodeconfig: 'AppiumNodeConfigAndroidWeb.json'
[debug] [Appium] Starting auto register thread for grid. Will try to register every 5000 ms.
[Appium] Appium REST http interface listener started on 0.0.0.0:4723
[debug] [Appium] Appium successfully registered with the grid on 192.168.56.1:4444
[HTTP] --> GET /wd/hub/status
[MJSONWP] Calling AppiumDriver.getStatus() with args: []
[MJSONWP] Responding to client with driver.getStatus() result: {"build":{"version":"1.5.0","revision":"e6f1500728e48f4be59bb4ca2b476198f
6559840"}}
```

Figure 6-15. *Console output for Appium Android node registration*

3. The Selenium Grid console output should look like Figure 6-16, and the Selenium console at http://192.168.56.1:4444/grid/console# should show the newly registered node with its configurations (Figure 6-17).

```
Shankars-MacBook-Pro:drivers sgarg$ java -Djava.net.preferIPv4Stack=false -jar selenium-server-standalone-2.53.1.jar -rol
e hub
22:48:25.789 INFO - Launching Selenium Grid hub
2016-09-11 22:48:26.528:INFO::main: Logging initialized @1087ms
22:48:26.540 INFO - Will listen on 4444
22:48:26.590 INFO - Will listen on 4444
2016-09-11 22:48:26.593:INFO:osjs.Server:main: jetty-9.2.z-SNAPSHOT
2016-09-11 22:48:26.623:INFO:osjsh.ContextHandler:main: Started o.s.j.s.ServletContextHandler@28ac3dc3{/,null,AVAILABLE}
2016-09-11 22:48:26.653:INFO:osjs.ServerConnector:main: Started ServerConnector@25bbf683{HTTP/1.1}{0.0.0.0:4444}
2016-09-11 22:48:26.654:INFO:osjs.Server:main: Started @1212ms
22:48:26.654 INFO - Nodes should register to http://192.168.56.1:4444/grid/register/
22:48:26.655 INFO - Selenium Grid hub is up and running
22:48:42.529 INFO - Registered a node http://127.0.0.1:4723
```

Figure 6-16. *Selenium Grid terminal output for Selenium Grid registering a new node*

Figure 6-17. *Selenium Grid Console: node details*

4. The server and node are ready, so now you need to make changes in your test case to redirect the test cases to Selenium Grid, instead of just going to the Appium server.

5. You will use the test case created in earlier chapters, `AppiumSampleTestCaseAndroidWeb`, and make the necessary changes to execute it using the Selenium Grid setup. Replace the line where you create the driver object with this new suggestion:

```
//Line to be replaced:
driver = new AndroidDriver(new
URL("http://127.0.0.1:4723/wd/hub"), caps);

//New Line to be added - Driver object with Grid
address
driver = new AndroidDriver(new
URL("http://192.168.56.1:4444/wd/hub"), caps);
```

6. Execute the program as explained in the previous chapters. Selenium Grid should receive the request, create a new session, and redirect the request appropriately (Figure 6-18).

```
23:01:34.050 INFO - Trying to create a new session on node http://127.0.0.1:4723
23:01:34.050 INFO - Trying to create a new session on test slot {browserName=chrome, maxInstances=1, version=5.1, platfor
m=ANDROID}
```

Figure 6-18. *Selenium Grid response to a new session*

The Appium node should receive the request (Figure 6-19), and the program should be executed appropriately (Figure 6-20).

```
[MJSONWP] Calling AppiumDriver.createSession() with args: [{"platformVersion":"5.0","browserName":"chrome","platformName":"Android","dev
iceName":"ANDROID","platform":"ANDROID"},null,null,null]
[Appium] Creating new AndroidDriver session
[Appium] Capabilities:
[Appium]   platformVersion: '5.0'
[Appium]   browserName: 'chrome'
[Appium]   platformName: 'Android'
[Appium]   deviceName: 'ANDROID'
[Appium]   platform: 'ANDROID'
[BaseDriver] The following capabilities were provided, but are not recognized by appium: platform.
[BaseDriver] Session created with session id: 74928be5-0aff-493a-920c-1fb0670864c4
[debug] [AndroidDriver] Getting Java version
[AndroidDriver] Java version is: 1.8.0_51
[AndroidDriver] We're going to run a Chrome-based session
[AndroidDriver] Chrome-type package and activity are com.android.chrome and com.google.android.apps.chrome.Main
[ADB] Checking whether adb is present
[ADB] Using adb from /Users/sgarg/Documents/Softwares/android/platform-tools/adb
[AndroidDriver] Retrieving device list
```

Figure 6-19. *Appium node console output for a new session*

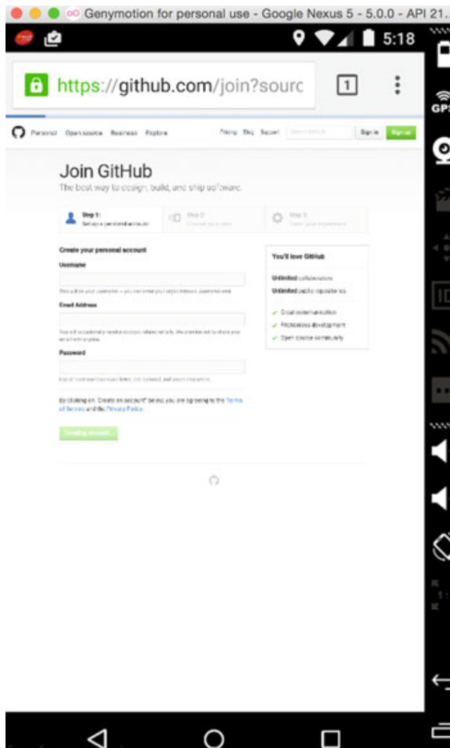


Figure 6-20. Android test case execution on an Android emulator

iOS

Now you need to create an Appium instance that will act as the iOS Safari node to the Selenium server. You will create a node configuration file called AppiumNodeConfigIOSWeb.json that will contain all the properties for this session.

1. Create a file called AppiumNodeConfigIOSWeb.json in the src/test/resources/AppiumConfig package in the AppiumRecipesBook project with the following content to create an Android iOS Safari instance:

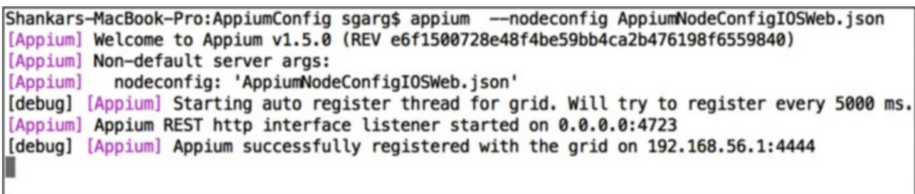
```
{
    "capabilities": [
                     {
      "maxInstances": 1,
      "browserName": "safari",
      "version": "9.2",
      "orientation": "LANDSCAPE",
      "platformName": "iOS",
```

```
                "platform": "MAC",
                "safariIgnoreFraudWarning": "true",
                "newCommandTimeout":999
                             }
                             ],
          "configuration":
          {
              "cleanUpCycle": 2000,
              "timeout": 300000,
              "browserTimeout": 60000,
                      "hub": "http://192.168.56.1:4444/grid/
                       register",
              "url":"http://127.0.0.1:4723/wd/hub",
              "host": "127.0.0.1",
              "port": 4723,
              "maxSession": 1,
              "register": true,
              "registerCycle": 5000,
              "hubPort": 4444,
              "hubHost": "192.168.56.1",
              "role": "node",
              "throwOnCapabilityNotPresent":"false"
          }
      }
```

2. To start an Appium node session, open a new terminal, cd to the `/src/test/resources/AppiumConfig` folder in the `AppiumRecipesBook` project, and start Appium with the following command:

   ```
   appium --nodeconfig AppiumNodeConfigIOSWeb.json
   ```

   The console output of the previous command should look like Figure 6-21.



```
Shankars-MacBook-Pro:AppiumConfig sgarg$ appium  --nodeconfig AppiumNodeConfigIOSWeb.json
[Appium] Welcome to Appium v1.5.0 (REV e6f1500728e48f4be59bb4ca2b476198f6559840)
[Appium] Non-default server args:
[Appium]   nodeconfig: 'AppiumNodeConfigIOSWeb.json'
[debug] [Appium] Starting auto register thread for grid. Will try to register every 5000 ms.
[Appium] Appium REST http interface listener started on 0.0.0.0:4723
[debug] [Appium] Appium successfully registered with the grid on 192.168.56.1:4444
```

Figure 6-21. *Console output for Appium iOS node registration*

3. The Selenium Grid console output should look like Figure 6-22, and the Selenium console at `http://192.168.56.1:4444/grid/console#` should show the newly registered node with its configurations (Figure 6-23).

```
23:09:36.040 INFO - Launching Selenium Grid hub
2016-09-11 23:09:36.555:INFO::main: Logging initialized @634ms
23:09:36.563 INFO - Will listen on 4444
23:09:36.593 INFO - Will listen on 4444
2016-09-11 23:09:36.595:INFO:osjs.Server:main: jetty-9.2.z-SNAPSHOT
2016-09-11 23:09:36.615:INFO:osjsh.ContextHandler:main: Started o.s.j.s.ServletContextHandler@28ac3dc3{/,null,AVAILABLE}
2016-09-11 23:09:36.633:INFO:osjs.ServerConnector:main: Started ServerConnector@25bbf683{HTTP/1.1}{0.0.0.0:4444}
2016-09-11 23:09:36.633:INFO:osjs.Server:main: Started @713ms
23:09:36.634 INFO - Nodes should register to http://192.168.56.1:4444/grid/register/
23:09:36.634 INFO - Selenium Grid hub is up and running
23:09:51.327 INFO - Registered a node http://127.0.0.1:4723
```

Figure 6-22. *Selenium Grid registering a new iOS node*

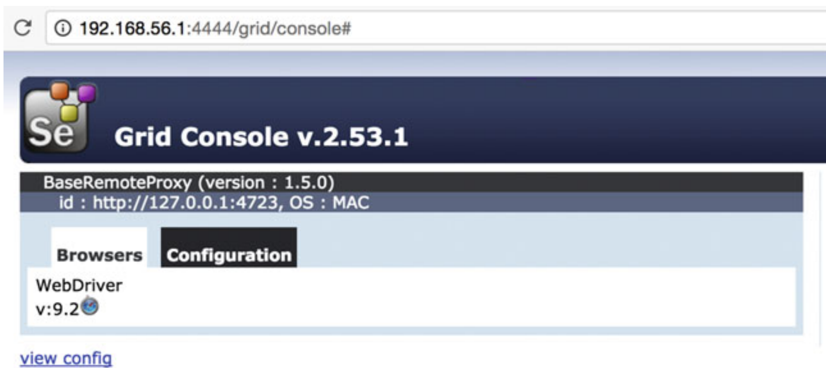


Figure 6-23. *Selenium Grid registering a new node*

4. The server and node are ready, so now you need to make changes in your test case to redirect the test cases to Selenium Grid, instead of just going to the Appium server.

5. You will use the test case created in earlier chapters, `AppiumSampleTestCaseIOSWeb`, and make the necessary changes to execute it using the Selenium Grid setup. Replace the line where you create the driver object with this new suggestion:

```
//Line to be replaced:
driver = new IOSDriver(new URL("http://127.0.0.1:4723/
wd/hub"), caps);
//New Line to be added - Driver object with Grid
address
driver =  new IOSDriver(new
URL("http://192.168.56.1:4444/wd/hub"), caps);
```

6. Execute the program as explained in the previous chapters. Selenium Grid should receive the request, create a new session, and redirect the request appropriately (Figure 6-24).

```
23:21:17.587 INFO - Got a request to create a new session: Capabilities [{platformVersion=9.2, browserName=safari, platfo
rmName=iOS, deviceName=iPhone 6, platform=iOS}]
```

Figure 6-24. Selenium Grid response to a new session

The Appium node should receive the request, and the program should be executed appropriately (Figure 6-25).

Figure 6-25. iOS test case execution

How It Works

Selenium Grid and the Appium node setting remain the same. The only difference is in the capabilities, which will set the browser to `safari` in the case of iOS and to `chrome` in the case of Android.

When a test case is executed, the Selenium server receives a request with certain desired capabilities, and then the server redirects that request to an Appium node session with the matching capabilities. So, in this recipe, those sessions would be for either Android Chrome or iOS Safari.

6-3. Appium with Selenium Grid for Two Android Sessions on the Same Machine

Problem

To reduce your infrastructure costs, you want to run multiple Android emulators on the same machine using Genymotion and use them as Appium nodes. For example, you want to run one Google Nexus 5 and one Google Nexus 10 on the same machine and use Nexus 5 only for mobile test cases and Nexus 10 only for tablet test cases.

Solution

In this recipe, you will set up Selenium Grid with two Android sessions, one on Nexus 5 and other on Nexus 10, and then execute the test case on the desired device.

For the Android solution presented here, you will use same grid setup as steps 1-3 in recipe 6-1.

1. Before you start with the Appium setup, knowing the device ID of both emulators is important (Figure 6-26). For this, run the following `adb` command on a terminal:

    ```
    adb devices
    ```

    Here 192.168.56.101:5555 is Nexus 5, and 192.168.56.102:5555 is Nexus 10.

```
Shankars-MacBook-Pro:AppiumConfig sgarg$ adb devices
List of devices attached
192.168.56.102:5555     device
192.168.56.101:5555     device
```

Figure 6-26. *List of Android devices*

2. Create the first config file called `AppiumNodeConfigAndroidNexus5.json` in the `src/test/resources/AppiumConfig` package in the `AppiumRecipesBook` project with the following content:

```
{
"capabilities":
      [
        {
              "maxInstances": 2,
              "browserName": "",
              "platform":"android",
              "version":"5.1",
              "deviceName": "192.168.56.101:5555"
        }
      ],
  "configuration":
  {
   "cleanUpCycle":2000,
   "timeout":30000,
   "proxy": "org.openqa.grid.selenium.proxy.
DefaultRemoteProxy",
   "url":"http://192.168.56.1:4723/wd/hub",
   "host": "192.168.56.1",
   "port": 4723,
   "maxSession": 1,
   "register": true,
   "registerCycle": 5000,
   "hubPort": 4444,
   "hubHost": "192.168.56.1"
  }
}
```

3. Create a second config file called `AppiumNodeConfigAndroidNexus10.json` in the `src/test/resources/AppiumConfig` package in the `AppiumRecipesBook` project with this content:

```
{
"capabilities":
      [
        {
              "maxInstances": 2,
              "browserName": "",
              "platform":"android",
              "version":"5.1",
              "deviceName": "192.168.56.102:5555"
```

```
        }
      ],
  "configuration":
  {
   "cleanUpCycle":2000,
   "timeout":30000,
   "proxy": "org.openqa.grid.selenium.proxy.
   DefaultRemoteProxy",
   "url":"http://192.168.56.1:4724/wd/hub",
   "host": "192.168.56.1",
   "port": 4724,
   "maxSession": 1,
   "register": true,
   "registerCycle": 5000,
   "hubPort": 4444,
   "hubHost": "192.168.56.1"
  }
}
```

4. To start an Appium node session for Nexus 5, open a new terminal, `cd` to the `/src/test/resources/AppiumConfig` folder in the `AppiumRecipesBook` project, and start Appium with the following command:

```
appium --nodeconfig AppiumNodeConfigAndroidNexus5.json
-p 4723
```

5. To start an Appium node session for Nexus 10, open a new terminal, `cd` to the `/src/test/resources/AppiumConfig` folder in the `AppiumRecipesBook` project, and start Appium with the following command:

```
appium --nodeconfig AppiumNodeConfigAndroidNexus10.json
-p 4724
```

■ **Note** Here the argument `-p` is important; `-p` is to specify the port number that a particular Appium session will use for its communication. Otherwise, both Appium sessions would want to use the same port and cause an error. This port number is the same as used for the `port` property in the `.json` file.

---

6. The Selenium console at `http://192.168.56.1:4444/grid/console#` should show the newly registered nodes (Figure 6-27).

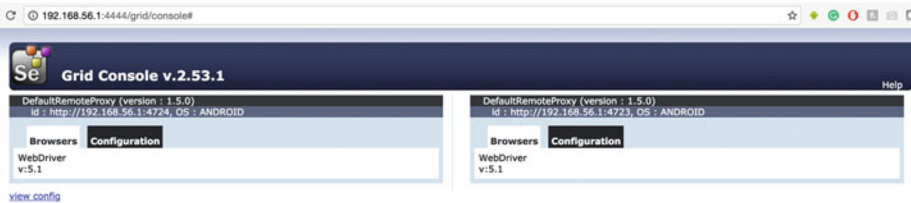


Figure 6-27. *Selenium Grid with two Android Appium sessions*

7. Make two copies of test case `AppiumSampleTestCaseAndroidNative` as `AppiumTestCaseNexus5` and `AppiumTestCaseNexus10`.

8. In `AppiumTestCaseNexus5`, use the following code for the Appium capability and keep everything else untouched:

```
// setting capabilities
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("platform", "ANDROID");
caps.setCapability("platformVersion", "5.0");
caps.setCapability("deviceName", "ANDROID");
caps.setCapability("browserName", "");
caps.setCapability("deviceName",
"192.168.56.101:5555");
```

9. In `AppiumTestCaseNexus10`, use the following code for the Appium capability and keep everything else untouched:

```
// setting capabilities
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("platform", "ANDROID");
caps.setCapability("platformVersion", "5.0");
caps.setCapability("deviceName", "ANDROID");
caps.setCapability("browserName", "");
caps.setCapability("deviceName",
"192.168.56.102:5555");
```

■ **Note** Here the `deviceName` capability is important because this will make sure that when you run one particular test case, the request that goes to Selenium Grid is for that particular device.

10. Execute the program `AppiumTestCaseNexus5` as explained in the previous chapters. Selenium Grid should receive the request, create a new session, and redirect the request to the Nexus 5 device only.
11. Execute the program `AppiumTestCaseNexus10` as explained in the previous chapters. Selenium Grid should receive the request, create a new session, and redirect the request to the Nexus 10 device only.

How It Works

When running multiple Android devices on the same machine and wanting to run specific test cases on specific devices, you need to bind the port number and device name together in the Appium config file as well as the test case.

In a config file, you specify to start an Appium session on device 1 on port 1, and in another config file, you specify to start an Appium session on device 2 on port 2. Then you specify one test case to run on device 1 and the second test case to run on device 2. When the test case is actually executed, Selenium Grid will send the request to the Appium session with the matching device name.

■ **Note** As of now, only one iOS session per machine can be started, so this scenario is not applicable to iOS.

CHAPTER 7

Executing Appium with Cloud Test Labs

In this chapter, you will learn to Execute:

- Appium on the Sauce Labs cloud
- Appium on the Testdroid cloud

In previous chapters, you learned to integrate Appium with Selenium Grid to execute test cases on an on-premise setup. In this chapter, you will learn to execute test cases on cloud test labs such as Sauce Labs and Testdroid.

Cloud test labs are subscription based (monthly, annual, and so on), which allows users to use a set of devices based on the subscription plan. The advantage of cloud test labs is that you don't need to maintain the devices and operating systems. You also don't need to worry about buying the latest versions in the market.

Although there are multiple cloud test labs available, this chapter will cover Sauce Labs and Testdroid. Sauce Labs provides emulators and simulators, but its real devices are expensive; Testdroid provides only real devices, and the costs are better comparatively. Both labs support Appium for the Android and iOS platforms.

7-1. Appium on the Sauce Labs Cloud

Problem

If you're familiar with web automation using Selenium, you are probably familiar with Sauce Labs. It's the official sponsor of both Selenium and Appium, so its integration with both these tools is obvious.

Test strategies that involve testing applications on various combinations of OS versions and devices like iOS 9.3.5 on iPhone 5s and iOS 9.1 on iPhone 6, and so on, will be best suited for Sauce Labs. You want to learn how to use Sauce Labs to execute Appium.

© Shankar Garg 2016
S. Garg, *Appium Recipes*, DOI 10.1007/978-1-4842-2418-2_7

Solution

In this recipe, you will register a new user for Sauce Labs and execute a native test case each for the Android and iOS native apps on the Sauce Labs cloud.

1. First you need to register at Sauce Labs to create an account. Go to `https://saucelabs.com/signup/trial` to create a free account.

2. After registration, you will get a verification e-mail. Verify the account and log in to Sauce Labs. You will be redirected to a dashboard (Figure 7-1). The left panel is the menu dashboard, and the right panel is the execution dashboard. For this recipe, you will use the Automated Tests dashboard.

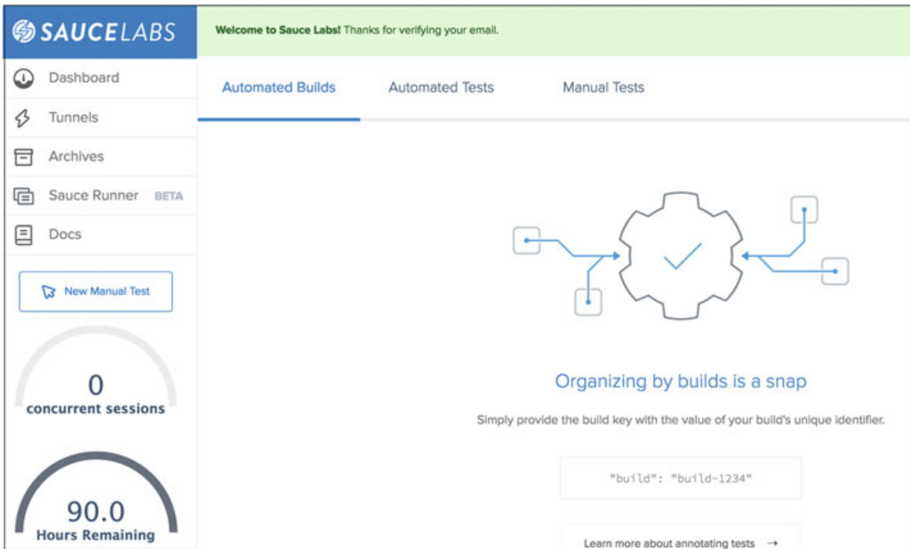


Figure 7-1. *Sauce Labs dashboard*

3. Now you need to write down the access key for your account. This access key acts as an identifier for your account.

    a. Scroll down in the left panel, click your name, and choose My Account from submenu (Figure 7-2).

    b. Go to the access key in the right panel. Click Show and write down the access key.

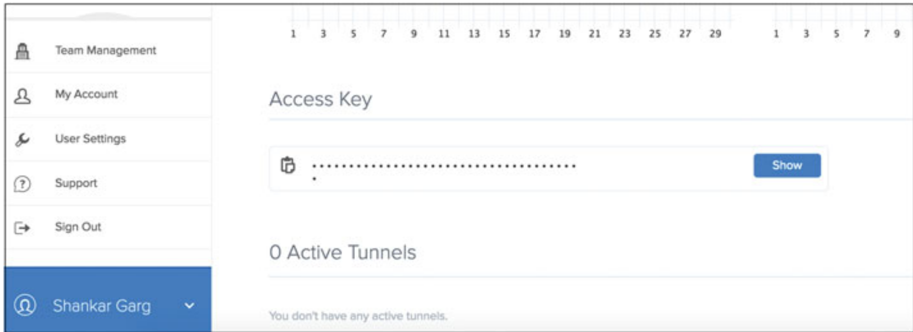


Figure 7-2. *Access key in the Sauce Labs dashboard*

4. Sauce Labs needs a reference to an application under test. There are two ways to do this; one is to upload the app to the cloud platform, which can be accessed publically by Sauce Labs, or you can upload the app to the Sauce Labs temporary storage. For this recipe, you will use the Sauce Labs temporary storage, so refer to `https://wiki.saucelabs.com/display/DOCS/Temporary+Storage+Methods` for all the required information.

Android

Follow these steps:

1. The command for uploading an app to Sauce Labs temporary storage is as follows:

```
curl -u <<YOUR_USERNAME>>:<<YOUR_ACCESS_KEY>> \
    -X POST \
    -H "Content-Type: application/octet-stream" \
    https://saucelabs.com/rest/v1/storage/<<YOUR_
    USERNAME>>/<<TEST_FILE_NAME>>?overwrite=true \
    --data-binary @<<PATH_TO_TEST_FILE>>
```

Here's what this code means:

`YOUR_USERNAME`: This is your Sauce Labs username.

`YOUR_ACCESS_KEY`: This is your Sauce Labs access key, noted in step 3.

`TEST_FILE_NAME`: This is the file name with which the file can be accessed on the Sauce Labs temporary storage.

`PATH_TO_TEST_FILE`: This is the absolute location of the file that you want to upload.

The file upload should return a message, as shown in Figure 7-3.

```
shankars-mbp:apps sgarg$ curl -u shankargarg1986:1 \
>     -X POST \
>     -H "Content-Type: application/octet-stream" \
>     https://saucelabs.com/rest/v1/storage/shankargarg1986/ApiDemos-debug.apk?overwrite=true \
>     --data-binary @ApiDemos-debug.apk
{"username": "shankargarg1986", "size": 3084877, "md5": "c651f3c857d9320dc2cbd3fdc560ca7d", "filename": "ApiDemos-debug.apk"}shankars-mbp:apps sgarg$
```

Figure 7-3. *Terminal output for Sauce Labs file upload*

■ **Note** The file extension for the iOS app is `.zip` (`.app` and `.ipa` files won't work).

Temporary storage is valid for only seven days; you will need to upload the app again after seven days.

Now you need to create an Appium test case that executes Android's `ApiDemos-debug.apk` file on Sauce Labs.

2. Create a file called `AppiumSauceLabsAndroid` in the `src/test/java/appium` package in the `AppiumRecipesBook` project with the following content:

```
package appium;

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

import java.net.MalformedURLException;
import java.net.URL;
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.support.
ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

public class AppiumSauceLabsAndroid {

public static void main(String[] args) throws
MalformedURLException, InterruptedException {
//Declaring WebDriver variables
AndroidDriver driver;
WebDriverWait wait;

// setting capabilities
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("platformName", "ANDROID");
caps.setCapability("platformVersion", "4.4");
```

```
caps.setCapability("deviceName", "Samsung Galaxy Nexus
Emulator");
caps.setCapability("browserName", "");
caps.setCapability("appiumVersion", "1.5.3");
caps.setCapability("app","sauce-storage:ApiDemos-debug.
apk");

// initializing driver object - Sauce Labs
// Replace credentials with yours
driver = new AndroidDriver(new URL("http://<<SauceLabs_
UserName>>:<<SauceLabs_accessID>>@ondemand.saucelabs.
com:80/wd/hub"), caps);

//initializing  waits
driver.manage().timeouts().implicitlyWait(10, TimeUnit.
SECONDS);
wait = new WebDriverWait(driver, 10);

// click on 'Accessibility' link
wait.until(ExpectedConditions.presenceOfElementLocated(
MobileBy.AccessibilityId("Accessibility")));
driver.findElement(MobileBy.AccessibilityId("Accessibil
ity")).click();
// click on 'Accessibility Node Querying' link
wait.until(ExpectedConditions.presenceOfElementLoc
ated(MobileBy.AccessibilityId("Accessibility Node
Querying")));
driver.findElement(MobileBy.
AccessibilityId("Accessibility Node Querying")).
click();
driver.navigate().back();
driver.navigate().back();

//using content-desc
driver.findElement(MobileBy.AndroidUIAutomator("descrip
tion(\"Accessibility\")")).click();
//close driver
driver.quit();

    }
}
```

3. Execute the program by right-clicking and selecting Run As ➤ Java Program.

4. Go to the Sauce Labs dashboard called Automated Tests. You should see one test case execution, as shown in Figure 7-4.

Figure 7-4. *Test case execution in the Sauce Labs dashboard*

5. Click the test case name in the dashboard to see the test case details (Figure 7-5).

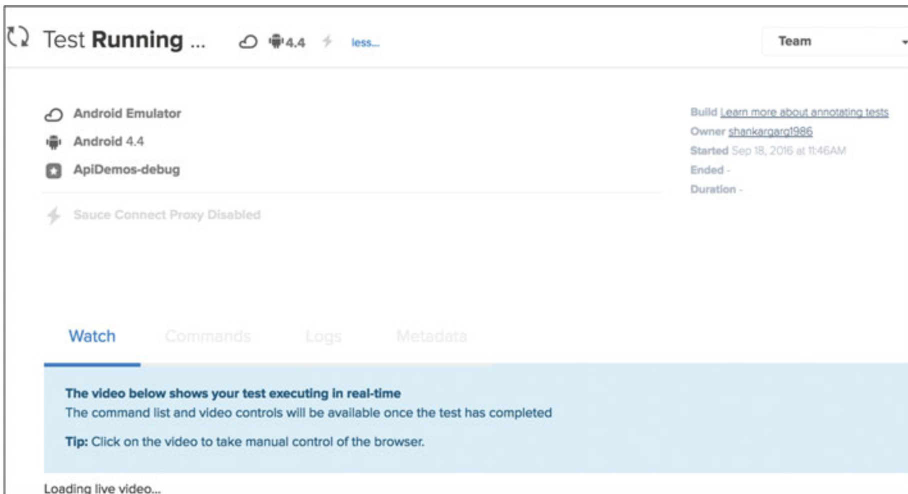


Figure 7-5. *Test case details while test case is executing*

6. Once the test case execution finishes, a video will be loaded, and you can view the video of the test case (Figure 7-6).

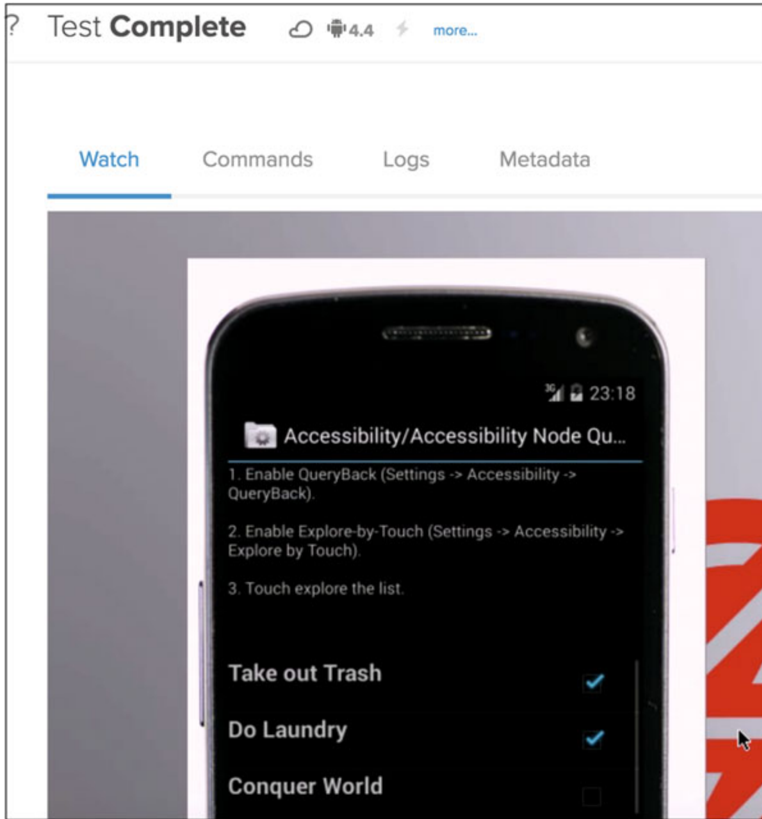


Figure 7-6. Test case details while test case is executing

iOS

Follow these steps:

1. The file upload for the iOS app should return a message like the one shown in Figure 7-7.

```
shankars-mbp:apps sgarg$ curl -u shankargarg1986:1[illegible]8    -X POST    -H "Content-Type: application/octet-stream"    https
://saucelabs.com/rest/v1/storage/shankargarg1986/TestApp.zip?overwrite=true    --data-binary @TestApp.zip
{"username": "shankargarg1986", "size": 36533, "md5": "19f4b6ac7c91b498908c4829d2d8c9b7", "filename": "TestApp.zip"}shankars-mbp:apps sgarg$
```

Figure 7-7. Terminal output for Sauce Labs file upload

■ **Note** The file extension for the iOS app is `.zip` (`.app` and `.ipa` files won't work).

Now you need to create an Appium test case that executes iOS's `TestApp.zip` file on Sauce Labs.

2. Create a file called `AppiumSauceLabsIOS` in the `src/test/java/appium` package in the `AppiumRecipesBook` project with the following content:

```
package appium;

import io.appium.java_client.MobileBy;
import io.appium.java_client.ios.IOSDriver;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.support.
ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

public class AppiumSauceLabsiOS {

public static void main(String[] args) throws
MalformedURLException, InterruptedException {
//Declaring WebDriver variables
IOSDriver driver;
WebDriverWait wait;

// setting capabilities
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("appiumVersion", "1.5.3");
caps.setCapability("deviceName","iPhone 6");
caps.setCapability("platformVersion","9.2");
caps.setCapability("platformName", "iOS");
caps.setCapability("browserName", "");
caps.setCapability("app","sauce-storage:TestApp.zip");

// initializing driver object - Sauce Labs
// Replace credentials with yours
driver =  new IOSDriver(new URL("http://<<SauceLabs_
UserName>>:<<SauceLabs_accessID>>@ondemand.saucelabs.
com:80/wd/hub"), caps);

// initializing waits
driver.manage().timeouts().implicitlyWait(10, TimeUnit.
SECONDS);
wait = new WebDriverWait(driver, 10);
```

```
//enter data in first text box
wait.until(ExpectedConditions.presenceOfElementLocated(
MobileBy.IosUIAutomation(".textFields()[0]")));
driver.findElement(MobileBy.IosUIAutomation(".
textFields()[0]")).sendKeys("1");

// enter data in second text box
wait.until(ExpectedConditions.presenceOfElementLocated(
MobileBy.IosUIAutomation(".textFields()[1]")));
driver.findElement(MobileBy.IosUIAutomation(".
textFields()[1]")).sendKeys("2");
// click on compute Sum Button driver.
findElement(MobileBy.IosUIAutomation(".buttons().firstW
ithPredicate(\"name=='ComputeSumButton'\")")).click();
// print value of '???' label
System.out.println(driver.findElement(MobileBy.
IosUIAutomation(".staticTexts().firstWithPredicate(\"na
me=='Answer'\")")).getText());

// close driver
driver.quit();

    }
}
```

3. Execute the program by right-clicking and selecting Run As ➤ Java Program.

4. Go to the Sauce Labs dashboard called Automated Tests. You should see one test case execution, as shown in Figure 7-8.

***Figure 7-8.** Test case execution in the Sauce Labs dashboard*

5. Click the test case name in the dashboard to see the test case details (Figure 7-9).

***Figure 7-9.** Test case details while test case is executing*

6. Once the test case execution finishes, a video will be loaded and you can view the video of the test case (Figure 7-10).

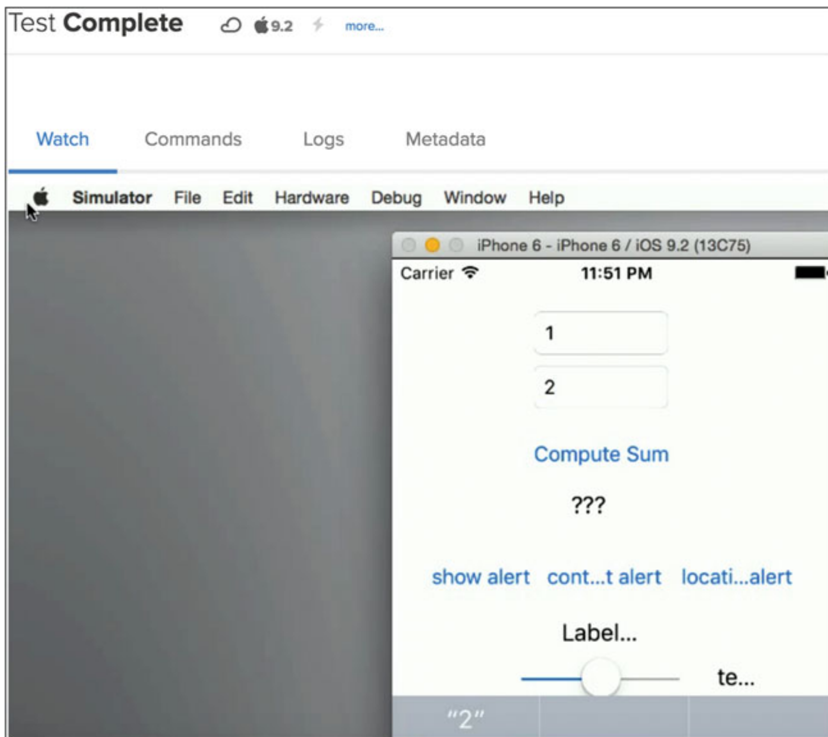


***Figure 7-10.** Test case details while test case is executing*

How It Works

To execute test cases on Sauce Labs, the whole setup can be divided into two parts.

- *Sauce Labs setup*: You need to register and create an account. Once you have created an account, you need to note the access key (an identifier for your account), and you need to upload the app that can be accessed by Sauce Labs. That's it.
- *Test case changes*: The beauty of Appium is that you don't need to make any changes in the test case, only in the desired capabilities.

Here's an example:

```
DesiredCapabilities caps = new DesiredCapabilities();
caps.setCapability("platformName", "ANDROID");
caps.setCapability("platformVersion", "4.4");
caps.setCapability("deviceName", "Samsung Galaxy Nexus
Emulator");
caps.setCapability("browserName", "");
caps.setCapability("appiumVersion", "1.5.3");
caps.setCapability("app","sauce-storage:ApiDemos-debug.apk");
```

For executing test cases on Sauce Labs, the main capabilities are as follows:

- `platformName`: Specify either `iOS` or `Android`.
- `platformVersion`: This is a specific version of the platform (for example, for iOS specify `8.0` or `9.3`; for Android, specify `5.0` or `6.0`).
- `deviceName`: Specify which device to use (for example, for iOS specify `iPhone 6`; or specify an actual device name such as `Samsung Galaxy S3` for Android).
- `app`: If you are executing a native app, then specify the location of the Sauce Labs temporary storage or the URL of the app somewhere on the Internet.
- `browserName`: If you are testing mobile web apps (for example, for iOS `Safari` or for Android `Chrome`), the browser name and app are mutually exclusive.
- `appiumVersion`: Specify which version of Appium to use for a particular execution; for example, 1.5.3 is the latest as of writing this book.

Here is how you define the Appium driver object:

```
driver =  new IOSDriver(new URL("http://<<SauceLabs_
UserName>>:<<SauceLabs_accessID>>@ondemand.saucelabs.com:80/wd/
hub"), caps);
```

Here you are redirecting the Appium execution to Sauce Labs instead of the local Appium instance.

You can find the entire list of desired capabilities for Sauce Labs here: `https://wiki.saucelabs.com/display/DOCS/Test+Configuration+Options`.

You can find the sample list of desired capabilities for Appium and Sauce Labs here: `https://wiki.saucelabs.com/display/DOCS/Examples+of+Test+Configuration+Options+for+Mobile+Native+Application+Tests`.

7-2. Appium on the Testdroid Cloud

Problem

Besides Sauce Labs, Testdroid is another cloud test lab. It provides real devices only, and you want to execute your test cases on real devices.

Solution

In this recipe, you will register a new user for Testdroid and will execute a native test case for Android native apps on the Testdroid cloud.

1. First you need to register at Testdroid to create an account. Go to `http://bitbar.com/testing/try-for-free/` to create a free account.

2. After registering, you will get a verification e-mail. Verify the account and log in to Testdroid. You will be redirected to a dashboard (Figure 7-11). The left panel is the menu dashboard, and the right panel is the execution dashboard.

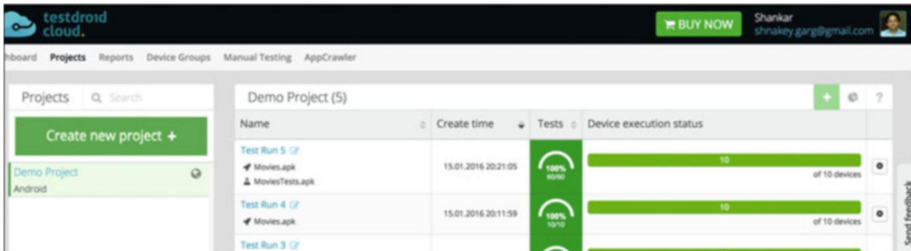


Figure 7-11. *Testdroid dashboard*

3. Go to Account Settings and check the subscription plan (Figure 7-12).

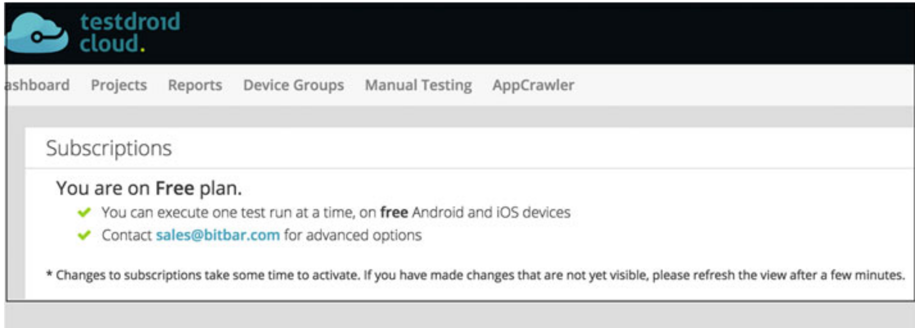


Figure 7-12. *Subscription plan*

4. Go to `https://cloud.testdroid.com/#service/devicegroups` and check the devices available for a trial subscription (Figures 7-13 and 7-14). Check the devices in the free Android category and the free iOS category and note the names. These will be used in the test case.

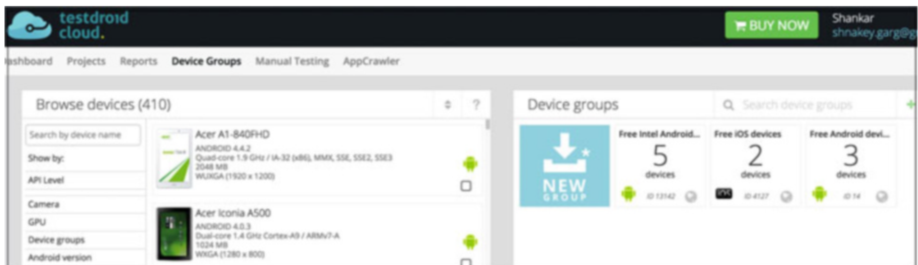


Figure 7-13. *Devices for trial plan*

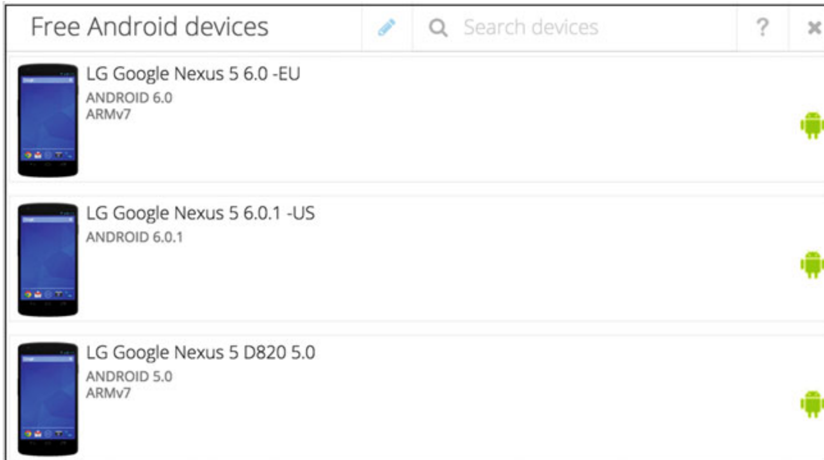


Figure 7-14. *Free Android devices*

5. Now you need to write down the access key for your account. This access key acts as an identifier for your account.

    a. In the Testdroid dashboard, mouse over your name and click "Account information."

    b. Go to the API key in the right panel. Write down the access key (Figure 7-15).

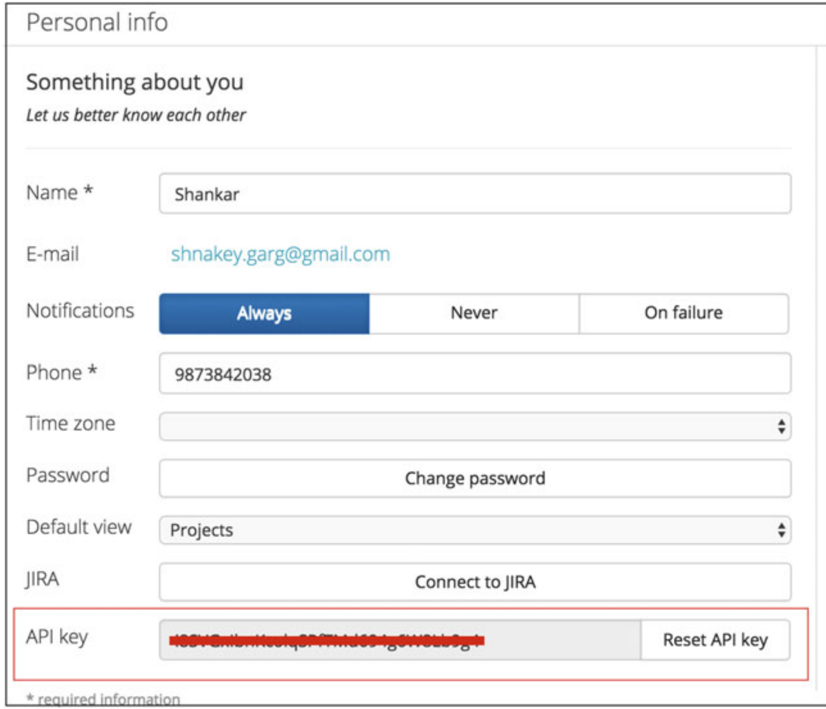


Figure 7-15. *API key in the Testdroid dashboard*

6. Go to the Testdroid dashboard, and in the left panel, create two projects (Figure 7-16).

    a. Create **Appiumbook** with a type of Appium Android.

    b. Create **AppiumBookios** with a type of Appium iOS.

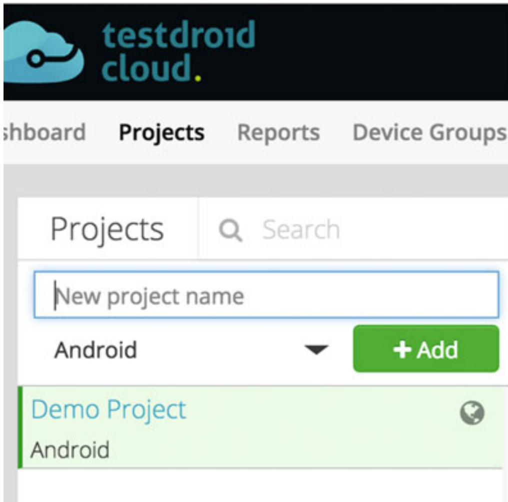


Figure 7-16. *Project in the Testdroid dashboard*

7. Testdroid needs a reference to the application under test. You are going to use the Testdroid temporary storage, so please refer to `http://testdroid.com/news/appium-testdroid-cloud-2` for all the required information.

Android

Follow these steps:

1. The command for uploading the app to Testdroid temporary storage is as follows:

```
curl -s --user <<testdroid_UserName>>:<<testdroid_
password>> -F myAppFile=@"<<absolute_File_path>>"
"http://appium.testdroid.com/upload"
```

   Here's what this code means:

   `TESTDROID_USERNAME`: This is the Testdroid username.

   `TESTDROID_PASSWORD`: This is the Testdroid password.

   `ABSOLUTE_FILE_PATH`: This is the absolute location of the file that you want to upload.

■ **Note** Down `myappfile` from upload response.

The file upload should return a message like Figure 7-17.

```
shankars-mbp:apps sgarg$ curl -s --user shnakey.garg@gmail.com: -F myAppFile=@"ApiDemos-debug.apk" "http://appium.testdroid.com/upload"
{"status":0,"sessionId":"af9de10f-cddf-4cae-a494-4c86a53e7552","value":{"message":"uploads successful","uploadCount":1,"rejectCount":0,"expiresIn
":1800,"uploads":{"myAppFile":"af9de10f-cddf-4cae-a494-4c86a53e7552/ApiDemos-debug.apk"},"rejects":{}}}shankars-mbp:apps sgarg$
```

Figure 7-17. *Terminal output for Testdroid file upload*

Now you need to create an Appium test case that executes Android's `ApiDemos-debug.apk` file on Testdroid.

2. Create a file called `AppiumTestDroidAndroid` in the `src/test/java/appium` package in the `AppiumRecipesBook` project with the following content:

```
package appium;

import io.appium.java_client.AppiumDriver;
import io.appium.java_client.MobileBy;
import io.appium.java_client.android.AndroidDriver;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.support.
ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

public class AppiumTestDroidAndroid {

public static void main(String[] args) throws
MalformedURLException, InterruptedException {
//Declaring WebDriver variables
AppiumDriver driver;
WebDriverWait wait;

// setting capabilities
DesiredCapabilities capabilities = new
DesiredCapabilities();

capabilities.setCapability("deviceName",
"AndroidDevice");
capabilities.setCapability("testdroid_target",
"Android");
capabilities.setCapability("testdroid_apiKey",
"<<API_Key>>");
capabilities.setCapability("testdroid_project",
"AppiumBook");
```

```
capabilities.setCapability("testdroid_testrun",
"Android Run 1");
capabilities.setCapability("testdroid_device", "LG
Google Nexus 5 6.0.1 -US");
capabilities.setCapability("testdroid_app", "af9de10f-
cddf-4cae-a494-4c86a53e7552/ApiDemos-debug.apk");
// initializing driver object - TestDroid
driver = new AndroidDriver(new URL("http://appium.
testdroid.com/wd/hub"), capabilities);

//initializing  waits
driver.manage().timeouts().implicitlyWait(10, TimeUnit.
SECONDS);
wait = new WebDriverWait(driver, 10);

// click on 'Accessibility' link
wait.until(ExpectedConditions.presenceOfElementLocated(
MobileBy.AccessibilityId("Accessibility")));
       driver.findElement(MobileBy.AccessibilityId("Acc
       essibility")).click();

// click on 'Accessibility Node Querying' link
wait.until(ExpectedConditions.presenceOfElementLoc
ated(MobileBy.AccessibilityId("Accessibility Node
Querying")));
driver.findElement(MobileBy.
AccessibilityId("Accessibility Node Querying")).
click();
driver.navigate().back();
driver.navigate().back();
//close driver
driver.quit();

    }
}
```

3. Execute the program by right-clicking and selecting Run As ➤ Java Program.

4. Go to the Testdroid dashboard and select the project AppiumBook. You should see one test case execution with the name Android Run 1, as shown in Figure 7-18.

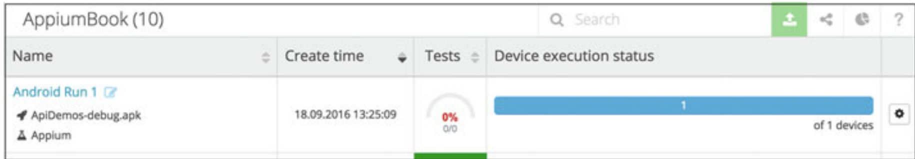


Figure 7-18. Test case execution in the Testdroid dashboard

5. Click the blue bar in the dashboard to see the test case details (Figure 7-19).

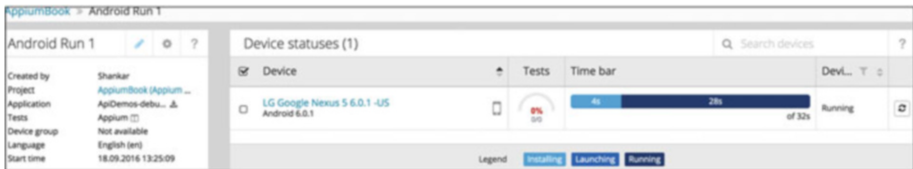


Figure 7-19. Test case details while test case is executing

6. Once the test case execution finishes, you will see the data shown in Figure 7-20.

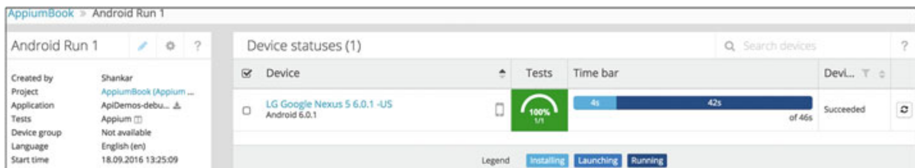


Figure 7-20. Test case details after test case has finished

7. You can click the blue card yet again to see the execution logs, such as the Appium logs, device logs, and performance dashboards (Figure 7-21).

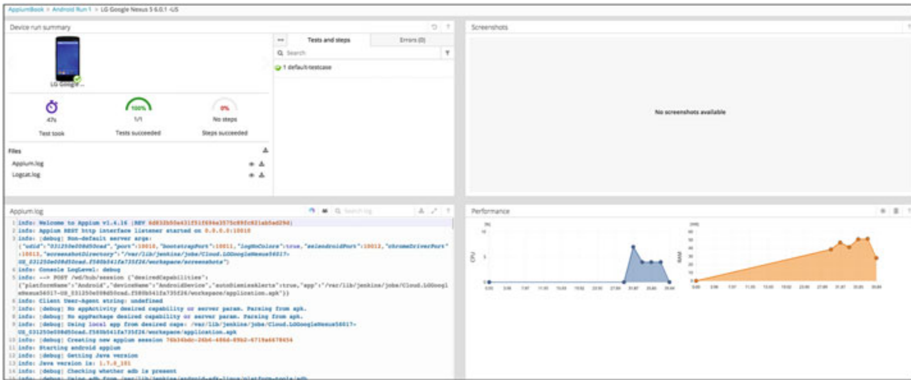


Figure 7-21. *Test case details and various logss*

iOS

Since Testdroid uses real devices for execution, the iOS app needs to be signed to be executed on real devices. For more information, visit `http://docs.testdroid.com/appium/environment/` and go to the iOS App Requirements section.

For this recipe, you will execute mobile web apps for iOS.

1. Create a file called `AppiumTestDroidIOSWeb` in the `src/test/java/appium` package in the `AppiumRecipesBook` project with the following content:

```
package appium;

import io.appium.java_client.AppiumDriver;
import io.appium.java_client.MobileBy;
import io.appium.java_client.ios.IOSDriver;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.support.
ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

public class AppiumTestDroidIOSWeb {
```

```
public static void main(String[] args) throws
MalformedURLException, InterruptedException {
// Declaring WebDriver variables
AppiumDriver driver;
WebDriverWait wait;

// setting capabilities
DesiredCapabilities capabilities = new
DesiredCapabilities();

capabilities.setCapability("deviceName", "iOS Phone");
capabilities.setCapability("testdroid_target",
"safari");
capabilities.setCapability("testdroid_apiKey",
"<<api_Key>>");
capabilities.setCapability("testdroid_project",
“AppiumBookIOS");
capabilities.setCapability("testdroid_testrun",
"Appium Run 3");

capabilities.setCapability("testdroid_device",
"iPhone 5c 7.0.4 A1532");
capabilities.setCapability("browserName", "safari");

// initializing driver object - TestDroid
driver = new IOSDriver(new URL("http://appium.
testdroid.com/wd/hub"), capabilities);

// initializing waits
driver.manage().timeouts().implicitlyWait(10, TimeUnit.
SECONDS);
wait = new WebDriverWait(driver, 10);

// open github URL
driver.get("https://github.com/");

// click Signup
wait.until(ExpectedConditions.
presenceOfElementLocated(By.linkText("Sign up for
GitHub")));
driver.findElement(By.linkText("Sign up for GitHub")).
click();

// click Create Account
wait.until(ExpectedConditions.presenceOfElementLocated(
MobileBy.id("signup_button")));
driver.findElement(MobileBy.id("signup_button")).click();

// close driver
driver.quit();

    }
}
```

2. Execute the program by right-clicking and selecting Run As ➤ Java Program.

3. Go to the Testdroid dashboard and select the project `AppiumBookios`. You should see one test case execution, as shown in Figure 7-22.

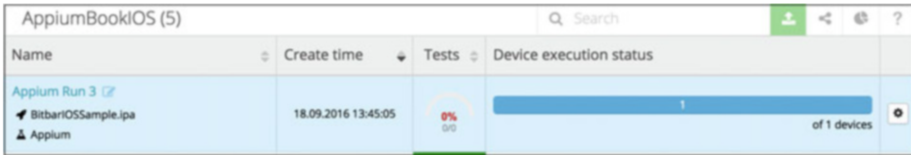


Figure 7-22. *Test case execution in the Testdroid dashboard*

4. Click the test case name in the dashboard and you will see data like in Figure 7-23.

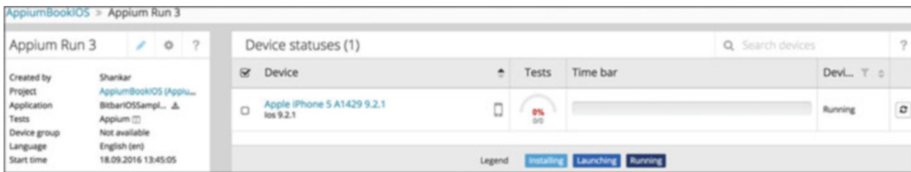


Figure 7-23. *Test case details while test case is executing*

5. Click the test case name in the dashboard and you will see data like Figure 7-24.

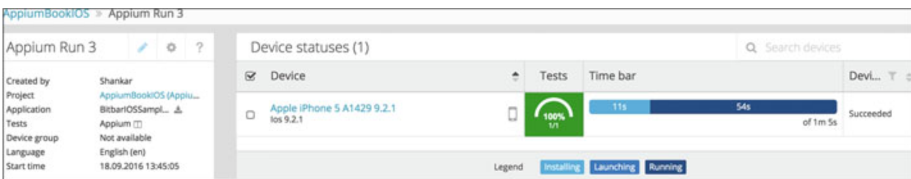


Figure 7-24. *Test case details while test case has finished*

6. You can click the blue card yet again to see the execution logs, such as the Appium logs, device logs, and performance dashboards (Figure 7-25).

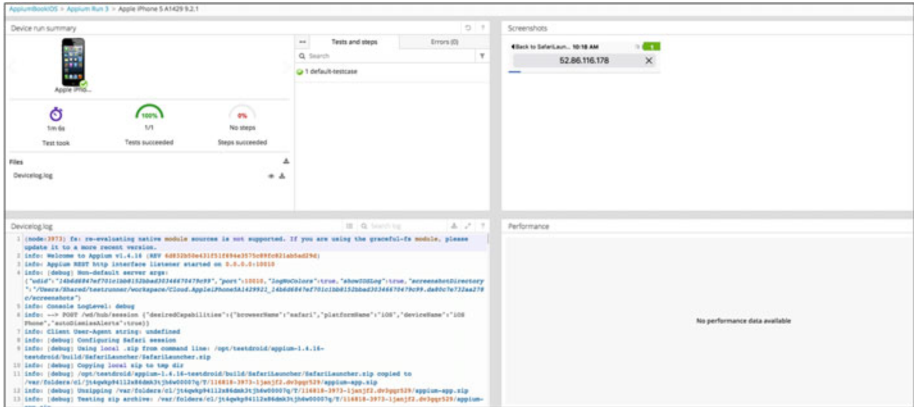


Figure 7-25. *Test case details and various logs*

How It Works

To execute test cases on Testdroid, the whole setup can be divided into two parts.

- *Testdroid setup*: You need to register and create an account. Once you have created an account, you need to note the API key (an identifier for your account), and you need to upload the app to be accessed by Testdroid.
- *Test case changes*: The beauty of Appium is that you don't need to make any changes in the test case, only in the desired capabilities.

  Here's an example:

```
DesiredCapabilities capabilities = new
DesiredCapabilities();
capabilities.setCapability("deviceName", "iOS Phone");
capabilities.setCapability("testdroid_target",
"safari");
capabilities.setCapability("testdroid_apiKey", "<<api_
Key>>");
capabilities.setCapability("testdroid_project",
"AppiumBookIOS");
capabilities.setCapability("testdroid_testrun", "Appium
Run 3");
capabilities.setCapability("testdroid_device", "iPhone
5c 7.0.4 A1532");
capabilities.setCapability("browserName", "safari");
```

For executing test cases on Testdroid, apart from the generic Appium capabilities, the following changes need to be made in the test case:

- deviceName*:* Use either AndroidDevice or iOS Phone.
- testdroid_target: Specify either iOS, Android, safari, or chrome.
- testdroid_device: Specify which device to use and get device names from https://cloud.testdroid.com/#service/devicegroups.
- testdroid_app*:* If you are executing a native app, then specify the location of the Testdroid temporary storage or specify the browser in the browserName capability.
- testdroid_apiKey: This is the API key for the Testdroid account.
- testdroid_project: This is the Testdroid project created for a particular platform.
- testdroid_testrun*:* This is the Testdroid test run to group different executions in an Appium project.

Here is how to define the Appium driver object:

```
driver = new IOSDriver(new URL("http://appium.testdroid.com/wd/hub"),
capabilities);
```

Here you specify to redirect the Appium execution to Testdroid instead of local execution.

You can find the list of desired capabilities for Testdroid here: http://help.testdroid.com/customer/portal/articles/1507074-testdroid_-desired-capabilities.

Index

A

Accessibility ID
  Android, 30–31
  elements, 33
  end users, 33
  generic locator strategies, 30
  iOS, 32
  name/content-desc attribute, 33
Android
  AppiumAndroid native node registration, 132
  Appium node session, 132, 135
  emulator, 135
  file Creation, 131
  native apps, 134
  node details, 133
  replacing, 134
  terminal output, 133
  test case execution, 135
Android mobile web elements
  ADB inspector, Google.com, 48
  build number item, about phone settings, 44
  Chrome ADB plug-in, 44–46, 48
  Developer options settings and USB debugging option, 45
  Discovering USB devices, 46
  screencast icon, 47
  USB debugging, 44
AndroidUIAutomator
  Android, 36
  API Demo Android application, 36
  problem, 35
  UISelector, 37
API Demo Android application, 30, 36
ApiDemos-debug.apk file, 158
Appium
  capabilities
    Android, 13
    client level (test case), 17
    coding, 16–17
    desired capabilities, 18
    GUI app, 12–13
    iOS, 12
    problem, 11
    server/client level, 11
    server level (GUI app/terminal), 17
    terminal, 14–15
  GUI (*see* Graphical user interface (GUI))
  mobile automation, 1
  NPM (*see* Node Package Manager (NPM))
AppiumBase file, 118
AppiumBase.java class, 116–117
AppiumDoctor, 5–6
AppiumDriverBase, 106–107
Appium Inspector, 22, 24
  attributes, 23–24
  GUI, 20
  hierarchy viewer, 23
  iOS, 20, 23
  Prelaunch Application check box, 21
  problem, 19
  properties, 22
AppiumNodeConfigAndroidNative.json, 131
AppiumNodeConfigAndroidWeb.json, 142
AppiumNodeConfigIOSNative.json, 136
AppiumNodeConfigIOSWeb.json, 145
AppiumRecipesBook project, 131, 136
AppiumSampleTestCaseAndroid, 134

© Shankar Garg 2016
S. Garg, *Appium Recipes*, DOI 10.1007/978-1-4842-2418-2

AppiumSampleTestCaseAndroidWeb, 144
AppiumSampleTestCaseIOS, 138
AppiumSampleTestCaseIOSWeb, 147
AppiumSauceLabsAndroid, 158–159
AppiumSauceLabsIOS, 162
AppiumTestDroidAndroid, 171–172
AppiumTestDroidIOSWeb, 174
Application programming
interface (API), 68
Automation frameworks, Appium
Android test case execution, 109
AppiumDriverBase, 106–107
CICD tools, 101
creation wizard, project
details, 103–104
integration, 110
Maven, and TestNG,
Android app, 101–102
mvn test, 109
pom.xml file, 105
project structure, 106
SampleTestCase, 108

B

@BeforeSuite, @BeforeTest, 110
Behavior-driven development (BDD), 101
AppiumBase file, 118
AppiumBase.java class, 117
AppiumCucumberPageObject
project, 113
console output, Appium Cucumber
project, 121
cucumber-java and
cucumber-junit, 112
Hooks.java class, 118
iOS app, 111
iOSTestApp.feature file, 114
pom.xml file, 111
problem, 110
RunCukeTest.java file, 114
src/test/resources package, 116
stepdefinition package, 113, 116
updation, 120

C

Chaining of actions, 84
Cloud test labs
Sauce Labs and Testdroid, 155
Continuous development (CD), 101
Continuous integration (CI) tool, 101
build number and timestamp, 126
console output, Appium project, 126
GitHub repository and
credentials, 122, 124
Jenkins dashboard, 123
pom.xml and Maven goal, 125
project building, 125
project description, 124
Source Code Management, 124
Cucumberoptions package, 114

D, E, F

Desired capabilities, 11, 18
Device network settings
Android, 99–100
data/airplane and Wi-Fi mode, 98
driver.SetConnection(), 100
Device orientation
Android, 92–93
LANDSCAPE/PORTRAIT, 93
mobile app development, 91
Rotate() function, 93
Domain-specific language (DSL), 122
Drag and drop elements
Android, 82–83
gaming apps, 82
methods, 84

G

GitHub, 122
Graphical user interface (GUI)
Android, 8
AppiumDoctor, 11
iOS, 9
problem, 6

H

Hooks.java class, 118
hubHost, 140
hubPort, 140
Hybrid apps
Android, 61–63
contexts types
native, 68
web view, 69
e-commerce, 61
iOS, 64–66

I

iOS mobile web elements
    properties, 38
    Safari
        Develop menu, 40
        Google.com, 42–43
        iOS simulator option, 41
        plug-in, 43
        Show Develop menu in menu bar, 39
        Web Inspector, 38
        web site opened in the simulator, 42
    Web Inspector, 43
iOS node
    Appium grid architecture, 140
    AppiumNodeConfigIOS Native.json, 136
    Appium node registration, 137
    AppiumSampleTest CaseIOS, 138
    config.json, 140
    configuration, 141
    creation, new session, 139
    grid setup, 136
    node details, 138
    Selenium Grid registering, 137
    /src/test/resources/AppiumConfig folder, 137
    test case execution, 139
iOSTestApp.feature file, 114
iOSTestAppSD.java, 116
iOSUIAutomation
    Appium Inspector, 33
    AppiumSampleTestCaseiOS class, 34
    compute Sum button, 34
    element-finding strategy, 35
    iOS, 34
    XPath expressions, 33, 35

J, K

JavaScript Object Notation (JSON), 11

L

Lock and unlock devices
    Android, 97–98
    driver.lockDevice(), 98
    driver.unlockDevice(), 98
    problem, 96

M

Mobile elements
    GitHub repository, 19
    principles, 19
Mobile web apps
    Android, 56–57
    browserName, 61
    Chrome and Google Play store apps, 56
    iOS, 58–60
    smartphones, 56
    web sites, 56
Mobile web automation
    Android
        Android test case execution, 145
        Appium Android node registration, 143
        Appium node console output, 144
        Appium node session, 143
        AppiumSampleTestCase AndroidWeb, 144
        creation, 142
        peplacing, 144
        Selenium Grid response, 143–144
        Selenium Grid terminal output, 143
    iOS
        AppiumSampleTestCaseIOSWeb, 147
        console output, 146
        creation, 145–146
        iOS test case execution, 148
        Selenium Grid registering, 147–148
    problem, 141
Mobility
    automating gestures, 77
    functions, 77
mvn test, 110

N, O

Native apps
    Android, 94–95
        ApiDemos-debug, 50–52
        sample test case, 52
    capabilities, 55
    driver.closeApp(), 96
    driver.installApp(), 96
    driver.removeapp(), 96
    installing, upgrading and deleting applications, 93

Native apps (*cont.*)
iOS
sample test case, 55
TestApp, 52–54
launch, close, install and remove, 96
mobile automation, 49–50
UI elements, 55
nodeconfig AppiumNodeConfigAndroid Nexus10.json, 151
Node Package Manager (NPM)
and node, 2
Appium server, 5
downloaded packages list, 3–4
problem, 1
server running, 5

P, Q

Page object model (POM), 118–119
pom.xml file, 110

R

Real devices
Android
API-Demos, 73
AppiumSampleTestCaseAndroid class, 73
build number setting, 70
Developer options setting, 71
PdaNet, 72
terminal, 73
USB debugging setting, 69, 72
emulators and simulators, 69
iOS
AppiumRecipesBook project, 74
coding, 75
implicit and explicit wait initialization, 74
provisioning profiles, 74
sample test case, 76
UDID, 74, 76
traditional mobile automation tools, 69
RunCukeTest.java file, 114

S

SampleTestCase, 108
Sauce Labs, 129
access key, 157
Android
AppiumSauceLabsAndroid, 158–159
file upload, 158
Java Program., 160
temporary storage, 157
test case execution, 160–161
Appium driver object, 165
Appium execution, 166
dashboard, 156
ios, 161–164
problem, 155
register, 156
setup, 165
temporary storage, 157
test case changes, 165
Selenium grid, 1
Android Appium sessions, 152
android devices, 149
Appium node session, 151
AppiumTestCaseNexus 5, 153
AppiumTestCaseNexus10, 152–153
creation, config file, 150–151
local infrastructure *vs.* cloud lab, 129
native app automation
Android and iOS platforms, 130
grid console, 131
problem, 129
terminal output, 130
problem, 149
Sauce Labs and Testdroid, 129
Software development kit (SDK), 25
/src/test/resources/AppiumConfig folder, 137
src/test/resources package, 106, 116
stepdefinition package, 113
Swiping and scrolling
Android, 84–85, 87
iOS, 87–90
mobility, 84
Swipe() function
attributes, 91
syntax, 90
web element, 84

T

Tap mobile elements
Android, 78–81
Appium, 78
tap(fingers, element, duration), 81

tap(fingers, x, y, duration), 81
TouchAction class, 81
web automation, 78
TestApp.zip file, 162
Testdroid cloud
account information, 168
Android
file upload, 171
temporary storage, 170
test case details, 173–174
API key, 168–169
Appiumbook, 169
AppiumBookios, 169
Appium driver object, 178
dashboard, 166
devices, trial plan, 167
free Android devices, 167–168
iOS, 174–176
problem, 166
register, 166
subscription plan, 167
Test case changes, 177
Testdroid setup, 177

U, V, W, X, Y, Z

UI Automator Viewer, 26
Appium
Inspector, 25
attributes, 29–30
default screen, 27
emulator, 30
problem, 25
properties, 28
steps, 25
User interface (UI), 7
built-in utility, 20
utils package, 116–117