Create an Enterprise-Level Test Automation Framework with Appium

Using Spring-Boot, Gradle, Junit, ALM Integration, and Custom Reports with TDD and BDD Support

—

Koushik Das

Apress®

Create an Enterprise-Level Test Automation Framework with Appium

Using Spring-Boot, Gradle, Junit, ALM Integration, and Custom Reports with TDD and BDD Support

Koushik Das

Apress®

Create an Enterprise-Level Test Automation Framework with Appium: Using Spring-Boot, Gradle, Junit, ALM Integration, and Custom Reports with TDD and BDD Support

Koushik Das
Miami, FL, USA

ISBN-13 (pbk): 978-1-4842-8196-3 ISBN-13 (electronic): 978-1-4842-8197-0
https://doi.org/10.1007/978-1-4842-8197-0

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Aaron Black
Development Editor: James Markham
Coordinating Editor: Jessica Vakili
Copyeditor: April Rondeau

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, email 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 booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles 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 Print and eBook Bulk Sales webpage at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on the GitHub repository: https://github.com/Apress/Create-an-Enterprise-Level-Test-Automation-Framework-with-Appium. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Table of Contents

About the Author

Koushik Das is an automation architect based in the United States and has over 18 years of experience with software development, manual testing, and test automation. He built automation frameworks for web and desktop applications using Shell script and Java prior to moving on to Selenium in 2013. He initially worked with TestNG and Maven before switching to Gradle and Junit in 2018. Besides Java, Koushik has worked with Kotlin for his more recent projects. As a senior test automation engineer and architect, he has been involved in several build-or-buy decisions on past projects. Koushik believes in leveraging the power of open source tools in test automation whenever possible and thinks that, given the proper handholding, anyone with beginner experience can graduate to an architect role.

Koushik can be reached on LinkedIn at `https://www.linkedin.com/in/daskoushik`.

About the Technical Reviewer

Vishwesh Ravi Shrimali graduated from BITS Pilani, where he studied mechanical engineering, in 2018. Since then, he has worked with BigVision LLC on deep learning and computer vision and was involved in creating official OpenCV AI courses. Currently, he is working at Mercedes-Benz Research and Development India Pvt. Ltd. He has a keen interest in programming and artificial intelligence and has applied that interest to mechanical engineering projects. He has also written multiple blogs on OpenCV and deep learning on LearnOpenCV, a leading blog on computer vision. He has also authored *Machine Learning for OpenCV* (2nd edition) by Packt. When he is not writing blogs or working on projects, he likes to go on long walks or play his acoustic guitar.

Introduction

Target Audience

This book is primarily written for software professionals in test automation with one to three years of experience and who have intermediate knowledge of Java. Prior project experience in mobile test automation is not required, but it is assumed that the reader knows about mobile test automation and concepts like Appium, Android, and iOS drivers. Basic knowledge of Cucumber Gherkin syntax is also assumed.

However, even recent engineering graduates who have programming knowledge and want to prepare themselves for a role in mobile test automation should refer to this book.

Mobile test automation is no doubt the technology of the moment, and test automation engineers are in great demand. This book is my attempt to help test engineers and computer science graduates to become pros in test automation. After reading this book, you should have full confidence in building new test automation frameworks.

How to Use This Book

Chapter 1 of this book gives a high-level overview of the automation framework. You should first go through this chapter to understand the technology stack, key features of the framework, and how this framework is different from some of the existing frameworks. I have included the framework diagram, which gives the complete architecture. You should refer to the framework after reading each subsequent chapter to understand how what you just read fits into the complete picture.

You should install the tools used in the framework on your personal computer and start coding while reading each chapter. That way, you will get hands-on experience and the feel of building a framework from scratch. I have added an "Automation Setup" section in the Appendix to which you can refer.

You need to use a sample .apk file for Android and a sample .ipa file for iOS. After writing a test suite, you need to run the test and generate a report. This is very important, as it will give you the confidence to take up the challenges of an architect role.

Credits and Acknowledgments

https://www.oodesign.com/interface-segregation-principle.html
https://www.tutorialsteacher.com/ioc/dependency-inversion-principle
https://github.com/extent-framework/extentreports-cucumber4-adapter
https://github.com/alonso05/ALM/blob/master/src/qc/rest/examples/AuthenticateLoginLogoutExample.java
https://gradle.org/maven-vs-gradle/
https://junit.org/junit5/
https://assertj.github.io/doc/
https://www.baeldung.com/spring-webclient-resttemplate
https://projectlombok.org/
https://docs.qameta.io/allure/
http://kong.github.io/unirest-java/
http://appium.io/
https://cucumber.io/docs/bdd/
https://spring.io/
https://www.baeldung.com/junit-testinstance-annotation
https://mkyong.com/java/java-how-to-send-email/
https://gradle.org/install/

Acronyms Used

TDD – Test-Driven Development
BDD – Behavior-Driven Development
DSL – Domain-Specific Language
QC – Quality Center
ALM – Application Lifecycle Management
OCR – Optical Character Recognition
FW - Framework
SIT – System Integration Testing
AT – Acceptance Testing
POJO – Plain Old Java Object
ADB – Android Debug Bridge
CRUD – Create, Read, Update, Delete
CRU – Create, Read, Update
NPM – Node Package Manager
JVM – Java Virtual Machine

Copyright and Disclaimer

CHAPTER 1

Automation Framework Overview

A test automation framework (Figure 1-1) establishes the toolset and reusable utility classes with which test engineers can build tests.
A framework needs to evolve as per the enterprise requirements while being stable enough to allow test engineers to focus on developing test scripts. All testing organizations involved in test automation follow a test automation framework built by architects. In this chapter, we will discuss the technology stack and key features of our automation framework.
We will also discuss the scripting strategy and coding standards followed throughout this book. Let's get started with the technology stack.

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_1

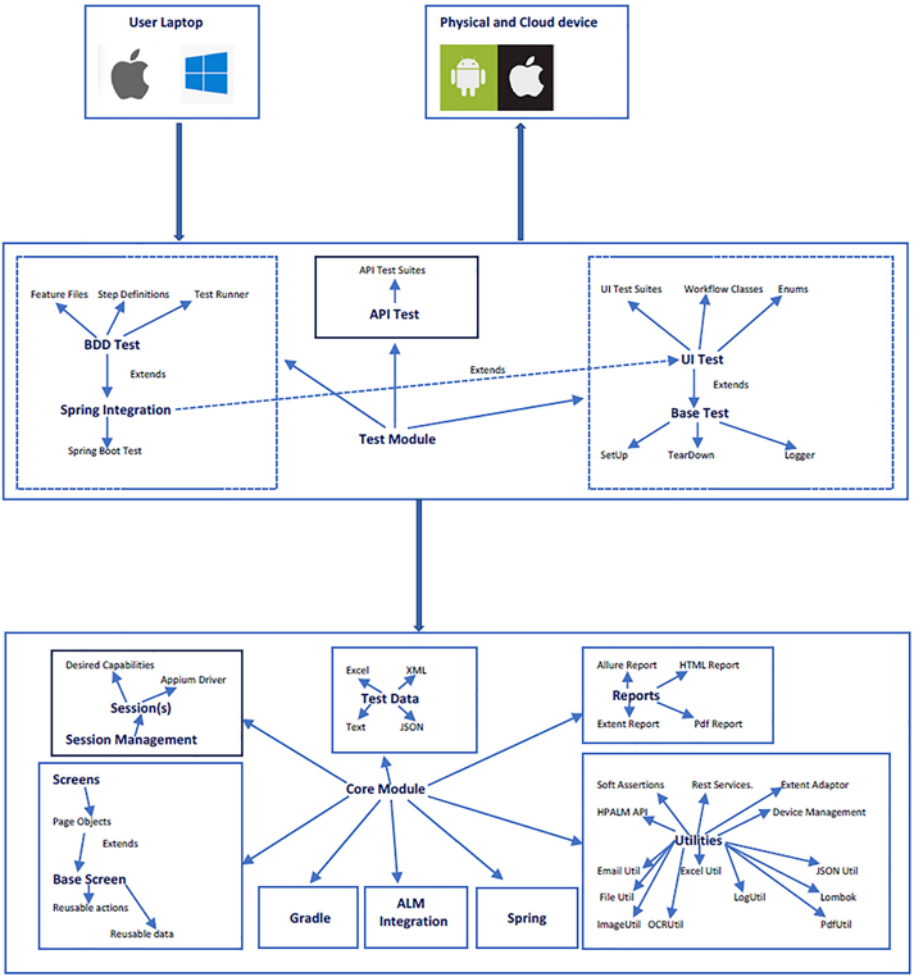


Figure 1-1. *Automation framework diagram*

Framework Technology Stack

We want to build this framework using the following technology stack:

- Gradle – build tool. Gradle was first released in 2007 and is currently distributed as open source software

under Apache License 2.0. Gradle uses DSL and is faster than Maven, which is XML based. For a detailed comparison with Maven, please refer to https://gradle.org/maven-vs-gradle/.

- JUnit5 – test runner. Junit5 is the current generation of Junit. Please refer to https://junit.org/junit5/ for a detailed user guide
- AssertJ – assertion/validation check. AssertJ provides highly readable soft assertions that are used throughout this book. Please refer to https://assertj.github.io/doc/ for a detailed overview.
- WebClient – REST API testing component. This book uses WebClient for REST API testing as it is asynchronous and non-blocking in nature, as compared to RestTemplate. For a detailed comparison, please refer to https://www.baeldung.com/spring-webclient-resttemplate.
- Lombok – reduction of boilerplate code. Project Lombok is a Java library that reduces boilerplate code, like writing getter and setter methods, constructors, and so on. For a detailed feature overview, please refer to https://projectlombok.org/.
- Allure – reporting. Allure is a widely used reporting tool, especially with TDD projects. Please refer to https://docs.qameta.io/allure/ for a detailed overview.
- HP ALM Rest API – integration with HPQC. Starting with ALM version 11, HP exposed some APIs that can be used to connect to HP QC. This book covers HP QC integration with ALM version 15 in an advanced topic.

- Unirest – integration with HPQC. I have used a Unirest-Java library for connecting to HP QC. Unirest is an open source, lightweight HTTP client library released under MIT license. For good documentation on Unirest, please refer to http://kong.github.io/unirest-java/.
- Appium – mobile application/web testing component. Appium is by far the most popular test automation tool for mobile-based applications. Appium is open source and platform independent. Please refer to http://appium.io/ for detailed documentation.
- Cucumber – BDD implementation. This book uses Cucumber for BDD implementation. Cucumber uses a logical language, Gherkin, to describe the test's scenarios and steps. For a detailed overview of Cucumber and Gherkin syntax, please refer to https://cucumber.io/docs/bdd/.
- Customized Cucumber Extent Adaptor – for implementation with Junit. The standard extent adaptor would not work for Junit as Junit does not have a default extent report listener, unlike TestNG. So, I have customized the adaptor to make it work with Junit. Also, a standard extent report does not print a data table. I have customized the adaptor for our purpose to print data table.

- Spring – dependency injection, reading properties file. Spring is a popular open source Java framework used extensively in building microservices. This book uses the Spring framework to leverage some of its annotations, like dependency injection, reading from properties file, and so on. For a detailed feature overview of Spring, please refer to https://spring.io/.

Framework Key Features

The following are the key features of our framework:

- Supports mobile native app automation for both iOS and Android
- Supports mobile web automation for both iOS and Android
- Supports web application automation (though it is not covered in this book)
- Supports web service automation for REST and SOAP (only REST is covered in this book)
- Supports SQL and no-SQL Databases
- Integrated with HP ALM through REST API
- Generates HTML, Allure report (TDD, BDD), and Extent report (BDD), and email notifications
- Supports multi-driver
- Supports parallel execution

Please refer to Table 1-1 for a feature comparison between our framework and most other frameworks.

Table 1-1. *Our Framework Compared to Traditional Non-Spring Frameworks*

Features	Our FW	BDD FW	TDD FW
Dependency injection through auto-wiring	Yes	No	No
Reading data from property file with Spring annotation	Yes	No	No
Removing boilerplate code (getter/setter methods in page objects) using Lombok	Yes	No	No
Supports BDD framework	Yes	Yes	No
Supports non-BDD framework without refactoring	Yes	No	Yes
Can be integrated with JIRA?	Yes	Yes	Yes
Can be integrated with HP-QC?	Yes	Yes	Yes
Uses Gradle as a build tool that is DSL based and faster	Yes	No	No
Uses fluent assertions (from AssertJ)	Yes	No	Yes
Supports parallel execution	Yes	Yes	Yes
Supports parameterized and ordered tests	Yes	Yes	Yes
Supports custom PDF reporting	Yes	No	No

We will discuss our scripting strategy and coding standard in the next two sections.

Scripting Strategy

A strong scripting strategy follows these guidelines:

- Test cases/methods should be tagged for smoke/regression/SIT/AT. This allows us to run a specific set of test cases/methods based on testing stage and requirements.

- Separate Gradle tasks should be created to execute specific groups of test cases. These tasks would correspond to the tags in the previous bullet point.
- Scripts that are for common workflows should be grouped in `Workflows` folder. In our framework, we would have a `ScreenNavigation.java` class (Listing 7-4) as our workflow class. This class would have specific navigation method(s) for each test suite.

Automation Coding Standards

Start by writing clean code as follows:

- Follow the builder pattern and use the Lombok plugin to avoid having boilerplate code. Use Project Lombok for building and maintaining POJOs. This helps to avoid redundant getters/setters.
- Use `@Step` annotation for Allure reports and `@DisplayName` annotation before every method. `@DisplayName` can be tagged to a requirement ID or test-case ID.
- Follow the Page Object model. Abstract element locators and methods to their respective pages. For object locators, use ID/accessibility ID/predicate. If these are not available, use others.
- Define methods with not more than three arguments. Functions with more than three arguments are heavy on implementation and difficult to maintain and refactor. A better alternative is to create higher-level methods or to create a class and pass an object as an argument to a method.

- Use comments whenever required and appropriately. Comments should accompany every test case and method and should describe what the algorithm is doing. Avoid unnecessary comments, as that reflects code smell.
- Use consistent style for coding. Use Command + Option + L for code alignment on a Mac when using IntelliJ editor. In Windows, use Ctrl + Alt + L.
- Use intention-revealing names for variables, methods, and classes. A meaningful name increases readability of the code and avoids code smell. The name must reflect the use and give context. So, the page class for app login should be named `AppLoginScreen` and not `GetStartedScreen`, which could mean many things!
- Use pronounceable names. Avoid tongue twisters and meaningless names.
- Don't add unnecessary context. Do not add meaningless prefixes or postfixes to class, variable, or method names. For example, `empName` is more appropriate than `empNameText`.
- Method names should say what they do. For example, `setEmpName` is more appropriate as a method name to set an employee name than `empName`.
- A method should do one thing, and one thing only. A method doing more than one thing violates SRP principle and is hard to maintain and refactor.

```
/**
* Bad:
* In the below example, finding mail status should be done in a
separate method, and that method should be called*/
public void sortMails (List<Mail> mails) {
                                for (Mail mail : mails) {
                                      MailStatus mailStatus
                                      = repository.
                                      findStatus(mail.getId());
                                       if (mailStatus.
                                       isActive()) {
                                            sort(mail);
} } }
```

Then maintain the following SOLID (Single-Open- Liskov-Interface-Dependency) design principles:

- **S**ingle Responsibility Principle. A class should only have a single responsibility, meaning it should have only one functional reason to change

```
/**
* Bad:
* In the below example, Employee class is supposed to have
methods and variables related to employee,
* however the class does more than one thing, which is it
prints the Org Name. It clearly violates the rule of SRP. */
  public class Employee {
      private String empCode;
      private String empName;
      private LocalDate joiningDate;
      public String getEmpCode () {return empCode; }
      public String getEmpName () {return empName; }
```

```
public LocalDate getJoiningDate () {return joiningDate; }
public void setEmpCode (String code) {this. empCode = code; }
public void setEmpName (String name) { this. empName = name; }
public void setJoiningDate (LocalDate date) { this. joiningDate
= date; }
public void printOrgName () {System.out.println("XXXX"); } }
```

- **O**pen/Closed Principle. "Software entities . . . should be open for extension but closed for modification." The entity defines the contract and can be a class or an interface. While the data and methods in the parent class or interface should be stable so as to not require frequent modifications, these should enable child classes or implementation classes to implement as per requirement. So, when a `Dog` class is extending a `Mammal` class, the changes should be limited to the `Dog` class and not impact the `Mammal` class.
- **L**iskov Substitution Principle. "Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program." At its heart, LSP is about interfaces and contracts as well as how to decide when to extend a class rather than use another strategy such as composition to achieve your goal.

  ```
  /**
  * Bad:
  * In the below example, ThreeDimension class
  extends the TwoDimension class,
  * however the derived class alters the behavior
  of the base class by introducing one more
  ```

```
parameter. It clearly violates the
rule of LSP. */
  public class TwoDimension {
      private String height;
      private String width;
      ...
public void addHeight (int x, int y) {... }
public void addWidth (int x, int y) { ... }
} }
public class ThreeDimension extends
TwoDimension {
public void addHeight (int x, int y,
int z) {... }
public void addWidth (int x, int y,
int z) { ... }
}
```

- **I**nterface Segregation Principle. "Many client-specific interfaces are better than one general-purpose interface." This means clients should not be forced to depend on interfaces they do not use. https://www.oodesign.com/interface-segregation-principle.html provides a good example.
- **D**ependency Inversion Principle. One should "depend upon abstractions, [not] concretions." This says high-level modules should not depend on low-level modules. Both should depend upon abstractions (interfaces) and be implemented through dependency injection. https://www.tutorialsteacher.com/ioc/dependency-inversion-principle provides a good example.

Use Functional Programming Over Imperative Approach

The Java 8 stream API enables us to use the multi-core power of processors without implementing multi-threading. Functional languages are declarative in nature and easier to write programs with. Choose streams and lambda functions when you can.

```
//Bad:
  public static int getSum () {
          double sum = 0;
          for (int i = 0; i < numbersList.size(); i++) { if
          (!numbersList.isEmpty()) {
          sum = sum + numbersList.get(i); } }
                                return sum; }
          //Good:
public static int getSum () { return numbersList.stream().
reduce(0.0, Double::sum); }
```

Summary

In this chapter, we discussed the framework technology stack, key framework features, scripting strategy, and coding standards. We discussed the use case for each component of our technology stack. We discussed the salient features of our framework and how it is different from many existing frameworks. Finally, we discussed the scripting and coding processes followed and why they are used. With this, we are ready to create the wireframe of our framework. We will learn how in the next chapter.

CHAPTER 1

Automation Framework Overview

A test automation framework (Figure 1-1) establishes the toolset and reusable utility classes with which test engineers can build tests.
A framework needs to evolve as per the enterprise requirements while being stable enough to allow test engineers to focus on developing test scripts. All testing organizations involved in test automation follow a test automation framework built by architects. In this chapter, we will discuss the technology stack and key features of our automation framework.
We will also discuss the scripting strategy and coding standards followed throughout this book. Let's get started with the technology stack.

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_1

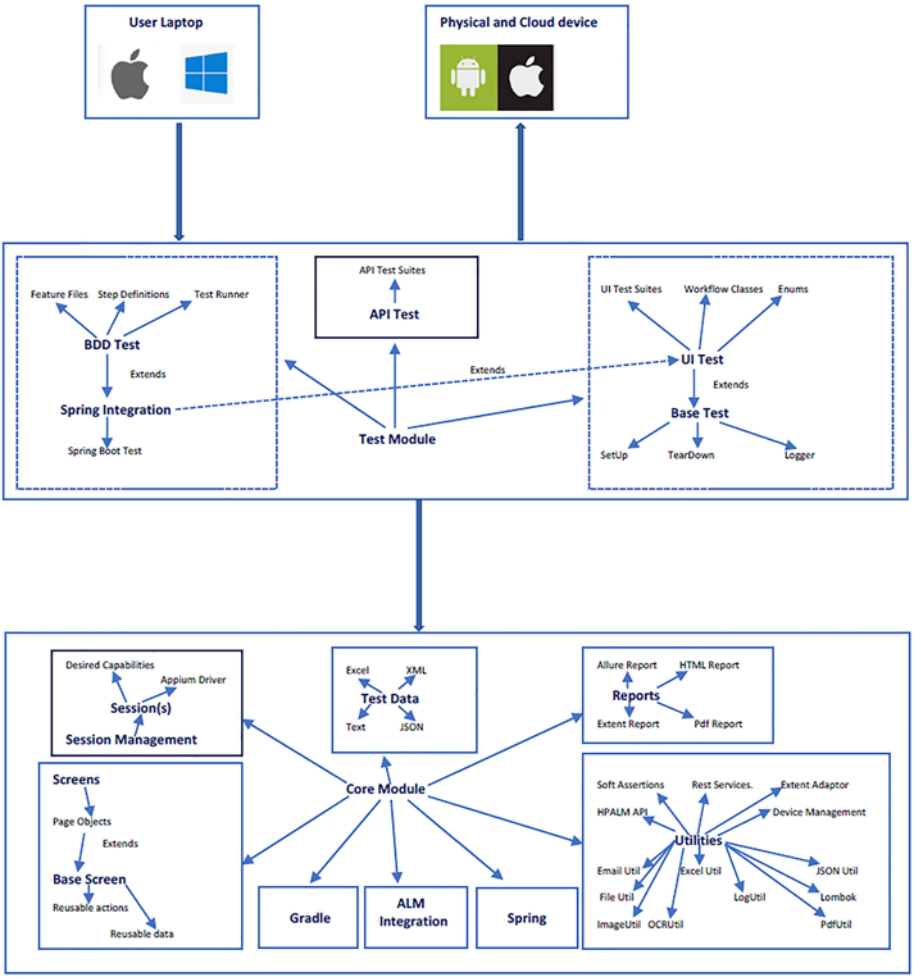


Figure 1-1. *Automation framework diagram*

Framework Technology Stack

We want to build this framework using the following technology stack:

- Gradle – build tool. Gradle was first released in 2007 and is currently distributed as open source software

under Apache License 2.0. Gradle uses DSL and is faster than Maven, which is XML based. For a detailed comparison with Maven, please refer to https://gradle.org/maven-vs-gradle/.

- JUnit5 – test runner. Junit5 is the current generation of Junit. Please refer to https://junit.org/junit5/ for a detailed user guide
- AssertJ – assertion/validation check. AssertJ provides highly readable soft assertions that are used throughout this book. Please refer to https://assertj.github.io/doc/ for a detailed overview.
- WebClient – REST API testing component. This book uses WebClient for REST API testing as it is asynchronous and non-blocking in nature, as compared to RestTemplate. For a detailed comparison, please refer to https://www.baeldung.com/spring-webclient-resttemplate.
- Lombok – reduction of boilerplate code. Project Lombok is a Java library that reduces boilerplate code, like writing getter and setter methods, constructors, and so on. For a detailed feature overview, please refer to https://projectlombok.org/.
- Allure – reporting. Allure is a widely used reporting tool, especially with TDD projects. Please refer to https://docs.qameta.io/allure/ for a detailed overview.
- HP ALM Rest API – integration with HPQC. Starting with ALM version 11, HP exposed some APIs that can be used to connect to HP QC. This book covers HP QC integration with ALM version 15 in an advanced topic.

- Unirest – integration with HPQC. I have used a Unirest-Java library for connecting to HP QC. Unirest is an open source, lightweight HTTP client library released under MIT license. For good documentation on Unirest, please refer to http://kong.github.io/unirest-java/.
- Appium – mobile application/web testing component. Appium is by far the most popular test automation tool for mobile-based applications. Appium is open source and platform independent. Please refer to http://appium.io/ for detailed documentation.
- Cucumber – BDD implementation. This book uses Cucumber for BDD implementation. Cucumber uses a logical language, Gherkin, to describe the test's scenarios and steps. For a detailed overview of Cucumber and Gherkin syntax, please refer to https://cucumber.io/docs/bdd/.
- Customized Cucumber Extent Adaptor – for implementation with Junit. The standard extent adaptor would not work for Junit as Junit does not have a default extent report listener, unlike TestNG. So, I have customized the adaptor to make it work with Junit. Also, a standard extent report does not print a data table. I have customized the adaptor for our purpose to print data table.

- Spring – dependency injection, reading properties file. Spring is a popular open source Java framework used extensively in building microservices. This book uses the Spring framework to leverage some of its annotations, like dependency injection, reading from properties file, and so on. For a detailed feature overview of Spring, please refer to https://spring.io/.

Framework Key Features

The following are the key features of our framework:

- Supports mobile native app automation for both iOS and Android
- Supports mobile web automation for both iOS and Android
- Supports web application automation (though it is not covered in this book)
- Supports web service automation for REST and SOAP (only REST is covered in this book)
- Supports SQL and no-SQL Databases
- Integrated with HP ALM through REST API
- Generates HTML, Allure report (TDD, BDD), and Extent report (BDD), and email notifications
- Supports multi-driver
- Supports parallel execution

Please refer to Table 1-1 for a feature comparison between our framework and most other frameworks.

Table 1-1. *Our Framework Compared to Traditional Non-Spring Frameworks*

Features	Our FW	BDD FW	TDD FW
Dependency injection through auto-wiring	Yes	No	No
Reading data from property file with Spring annotation	Yes	No	No
Removing boilerplate code (getter/setter methods in page objects) using Lombok	Yes	No	No
Supports BDD framework	Yes	Yes	No
Supports non-BDD framework without refactoring	Yes	No	Yes
Can be integrated with JIRA?	Yes	Yes	Yes
Can be integrated with HP-QC?	Yes	Yes	Yes
Uses Gradle as a build tool that is DSL based and faster	Yes	No	No
Uses fluent assertions (from AssertJ)	Yes	No	Yes
Supports parallel execution	Yes	Yes	Yes
Supports parameterized and ordered tests	Yes	Yes	Yes
Supports custom PDF reporting	Yes	No	No

We will discuss our scripting strategy and coding standard in the next two sections.

Scripting Strategy

A strong scripting strategy follows these guidelines:

- Test cases/methods should be tagged for smoke/regression/SIT/AT. This allows us to run a specific set of test cases/methods based on testing stage and requirements.

- Separate Gradle tasks should be created to execute specific groups of test cases. These tasks would correspond to the tags in the previous bullet point.
- Scripts that are for common workflows should be grouped in `Workflows` folder. In our framework, we would have a `ScreenNavigation.java` class (Listing 7-4) as our workflow class. This class would have specific navigation method(s) for each test suite.

Automation Coding Standards

Start by writing clean code as follows:

- Follow the builder pattern and use the Lombok plugin to avoid having boilerplate code. Use Project Lombok for building and maintaining POJOs. This helps to avoid redundant getters/setters.
- Use `@Step` annotation for Allure reports and `@DisplayName` annotation before every method. `@DisplayName` can be tagged to a requirement ID or test-case ID.
- Follow the Page Object model. Abstract element locators and methods to their respective pages. For object locators, use ID/accessibility ID/predicate. If these are not available, use others.
- Define methods with not more than three arguments. Functions with more than three arguments are heavy on implementation and difficult to maintain and refactor. A better alternative is to create higher-level methods or to create a class and pass an object as an argument to a method.

- Use comments whenever required and appropriately. Comments should accompany every test case and method and should describe what the algorithm is doing. Avoid unnecessary comments, as that reflects code smell.
- Use consistent style for coding. Use Command + Option + L for code alignment on a Mac when using IntelliJ editor. In Windows, use Ctrl + Alt + L.
- Use intention-revealing names for variables, methods, and classes. A meaningful name increases readability of the code and avoids code smell. The name must reflect the use and give context. So, the page class for app login should be named `AppLoginScreen` and not `GetStartedScreen`, which could mean many things!
- Use pronounceable names. Avoid tongue twisters and meaningless names.
- Don't add unnecessary context. Do not add meaningless prefixes or postfixes to class, variable, or method names. For example, `empName` is more appropriate than `empNameText`.
- Method names should say what they do. For example, `setEmpName` is more appropriate as a method name to set an employee name than `empName`.
- A method should do one thing, and one thing only. A method doing more than one thing violates SRP principle and is hard to maintain and refactor.

```
/**
* Bad:
* In the below example, finding mail status should be done in a
separate method, and that method should be called*/
public void sortMails (List<Mail> mails) {
                                for (Mail mail : mails) {
                                      MailStatus mailStatus
                                      = repository.
                                      findStatus(mail.getId());
                                       if (mailStatus.
                                       isActive()) {
                                            sort(mail);
} } }
```

Then maintain the following SOLID (Single-Open- Liskov-Interface-Dependency) design principles:

- **S**ingle Responsibility Principle. A class should only have a single responsibility, meaning it should have only one functional reason to change

```
/**
* Bad:
* In the below example, Employee class is supposed to have
methods and variables related to employee,
* however the class does more than one thing, which is it
prints the Org Name. It clearly violates the rule of SRP. */
  public class Employee {
      private String empCode;
      private String empName;
      private LocalDate joiningDate;
      public String getEmpCode () {return empCode; }
      public String getEmpName () {return empName; }
```

```
public LocalDate getJoiningDate () {return joiningDate; }
public void setEmpCode (String code) {this. empCode = code; }
public void setEmpName (String name) { this. empName = name; }
public void setJoiningDate (LocalDate date) { this. joiningDate
= date; }
public void printOrgName () {System.out.println("XXXX"); } }
```

- **O**pen/Closed Principle. "Software entities . . . should be open for extension but closed for modification." The entity defines the contract and can be a class or an interface. While the data and methods in the parent class or interface should be stable so as to not require frequent modifications, these should enable child classes or implementation classes to implement as per requirement. So, when a `Dog` class is extending a `Mammal` class, the changes should be limited to the `Dog` class and not impact the `Mammal` class.
- **L**iskov Substitution Principle. "Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program." At its heart, LSP is about interfaces and contracts as well as how to decide when to extend a class rather than use another strategy such as composition to achieve your goal.

  ```
  /**
  * Bad:
  * In the below example, ThreeDimension class
  extends the TwoDimension class,
  * however the derived class alters the behavior
  of the base class by introducing one more
  ```

```
parameter. It clearly violates the
rule of LSP. */
  public class TwoDimension {
      private String height;
      private String width;
      ...
public void addHeight (int x, int y) {... }
public void addWidth (int x, int y) { ... }
} }
public class ThreeDimension extends
TwoDimension {
public void addHeight (int x, int y,
int z) {... }
public void addWidth (int x, int y,
int z) { ... }
}
```

- **I**nterface Segregation Principle. "Many client-specific interfaces are better than one general-purpose interface." This means clients should not be forced to depend on interfaces they do not use. https://www.oodesign.com/interface-segregation-principle.html provides a good example.
- **D**ependency Inversion Principle. One should "depend upon abstractions, [not] concretions." This says high-level modules should not depend on low-level modules. Both should depend upon abstractions (interfaces) and be implemented through dependency injection. https://www.tutorialsteacher.com/ioc/dependency-inversion-principle provides a good example.

Use Functional Programming Over Imperative Approach

The Java 8 stream API enables us to use the multi-core power of processors without implementing multi-threading. Functional languages are declarative in nature and easier to write programs with. Choose streams and lambda functions when you can.

```
//Bad:
  public static int getSum () {
          double sum = 0;
          for (int i = 0; i < numbersList.size(); i++) { if
          (!numbersList.isEmpty()) {
          sum = sum + numbersList.get(i); } }
                                return sum; }
          //Good:
public static int getSum () { return numbersList.stream().
reduce(0.0, Double::sum); }
```

Summary

In this chapter, we discussed the framework technology stack, key framework features, scripting strategy, and coding standards. We discussed the use case for each component of our technology stack. We discussed the salient features of our framework and how it is different from many existing frameworks. Finally, we discussed the scripting and coding processes followed and why they are used. With this, we are ready to create the wireframe of our framework. We will learn how in the next chapter.

CHAPTER 3

Configuring Gradle

In the previous chapter, we created the wireframe of our framework with Spring-Boot and set up the initial folder structure. In this chapter, we will prepare a few Gradle configuration files for dependency management. We will also define the Gradle tasks that will be used for our test execution. Let's get started with the `build.gradle` file.

Preparing build.gradle

Let's open `build.gradle` in the editor. It looks like Figure 3-1 at this stage.

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_3

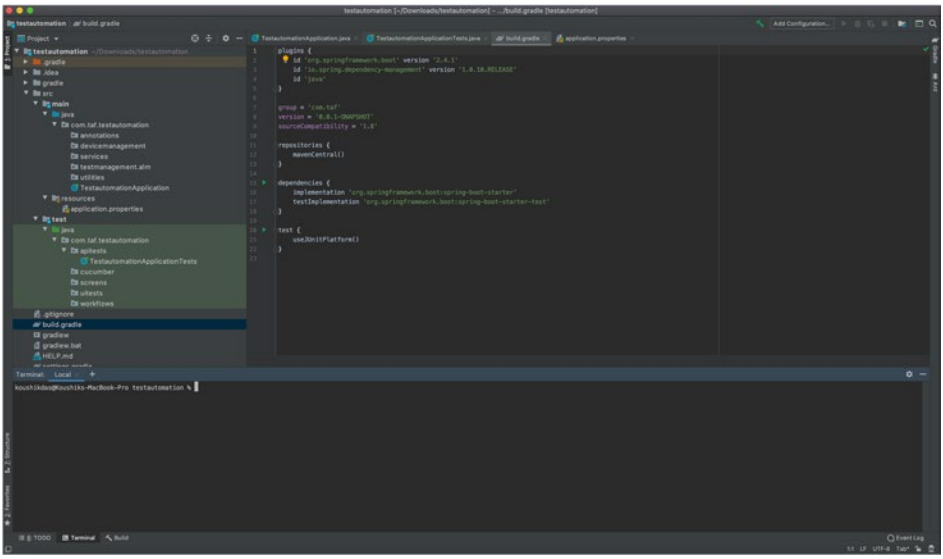


Figure 3-1. *build.gradle with initial Spring-Boot dependencies*

As you can see, it has only default Spring-Boot dependencies. So, we now add dependencies based on our needs. I will add the dependencies, as shown in Listing 3-1, and then explain each one.

Listing 3-1. build.gradle with Dependencies

```
plugins {
   id 'org.springframework.boot' version '2.4.1'
   id 'io.spring.dependency-management' version '1.0.10.RELEASE'
   id 'io.franzbecker.gradle-lombok' version '3.1.0'
   id 'io.qameta.allure' version '2.8.1'
   id 'java'
}
```

```
group = 'com.taf'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '1.8'

allure {
    version = '2.12.1'
    downloadLinkFormat = 'https://dl.bintray.com/qameta/
    maven/io/qameta/allure/allure-commandline/%s/allure-
    commandline-%<s.zip'
    useJUnit5 {
        version = '2.12.1'
    }
    autoconfigure = false
    aspectjweaver = true
    aspectjVersion = "${aspectj_version}"
}

repositories {
    jcenter()
    mavenLocal()
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-
    boot-starter'
    testImplementation('org.springframework.boot:spring-boot-
    starter-test') {
        exclude group: 'org.junit.vintage', module: 'junit-
        vintage-engine'
    }
    dependencies {
```

```
        compile "com.aventstack:extentreports-cucumber4-
        adapter:${report_version}"
    }

    compile "com.google.code.gson:gson:${gson_version}"
    compile "org.springframework.boot:spring-boot-starter-
    webflux:${spring_boot_version}"
    compile group: 'io.cucumber', name: 'cucumber-spring',
    version: '5.6.0'
    compile group: 'io.cucumber', name: 'cucumber-junit-
    platform-engine', version: '5.6.0'
    compile group: 'javax.xml.bind', name: 'jaxb-api',
    version: '2.3.1'
    compile group: 'javax.activation', name: 'activation',
    version: '1.1.1'
    compile group: 'org.glassfish.jaxb', name: 'jaxb-runtime',
    version: '2.3.1'
    compile group: 'com.aventstack', name: 'extentreports',
    version: '4.1.5'
    compile group: 'org.mongodb', name: 'mongo-java-driver',
    version: '3.12.3'
    compile group: 'com.sun.mail', name: 'javax.mail',
    version: '1.6.2'
    compile group: 'io.cucumber', name: 'cucumber-core',
    version: '5.6.0'
    compile group: 'org.apache.poi', name: 'poi',
    version: '4.1.2'
    compile group: 'org.apache.poi', name: 'poi-ooxml',
    version: '4.1.2'
    compile group: 'net.sourceforge.tess4j', name: 'tess4j',
    version: '4.5.3'
```

```
    compile group: 'com.itextpdf', name: 'itextpdf', version:
    '5.5.13.2'
    compile group: 'com.konghq', name: 'unirest-java', version:
    '3.11.01'

    testCompile("io.github.bonigarcia:webdrivermanager:3.7.1")
    testImplementation 'io.cucumber:cucumber-java:5.6.0'
    testImplementation 'io.cucumber:cucumber-junit:5.6.0'
    testImplementation "io.qameta.allure:allure-cucumber5-jvm:${
    allurePluginVersion}"

    implementation group: 'org.junit.jupiter', name: 'junit-
    jupiter', version: "${junit_jupiter_version}"
    implementation "org.assertj:assertj-core:${assertj_version}"
    implementation "org.junit.jupiter:junit-jupiter-
    params:${junit_jupiter_params_version}"
    implementation group: 'org.junit.jupiter', name: 'junit-
    jupiter-api', version: "${junit_jupiter_api_version}"
    implementation "org.junit.jupiter:junit-jupiter-
    engine:${junit_jupiter_engine_version}"
    implementation "io.appium:java-client:${appium_version}"
    implementation "io.qameta.allure:allure-junit5:${allure_
    junit_version}"
    implementation group: 'io.qameta.allure', name: 'allure-
    java-commons', version: "${allure_version}"
}
```

In the plugin section, in addition to the default Spring-Boot, I have also added the Lombok and Allure plugins. As I mentioned in Chapter 1, Project Lombok is an open source library used to reduce boilerplate code in Java. We also need AspectJ in order to use the Allure step annotations that we might use in various page object methods. The autoconfigure flag, when enabled, makes the plugin automatically add related Allure

dependencies like `aspectjweaver`, `allure java`, and so on. In our case, we set the autoconfigure flag to false, as we are specifying these dependencies ourselves. `Jcenter()` is a superset of `mavenCentral()`. `Jcenter()`, `mavenCentral()`, and `mavenLocal()` are the aliases used in Gradle when we add Maven repositories to our Gradle project.

In the dependencies section, I have excluded `org.junit.vintage`, as we are using Junit5. The `cucumber4-adapter` is for creating an extent report in BDD (we will cover this in Chapter 11), while we use the Gson library for converting JSON to `Map` objects in JsonUtil (Chapter 9) and vice versa for converting `JsonElement` response objects to JSON (Chapter 16).

The dependencies `cucumber-spring`, `cucumber-junit`, `cucumber-java`, and `cucumber-core` are required for Cucumber integration. `Jaxb-api` and `jaxb-runtime` are required to read or create XML files. These will be required for QC integration (Chapter 18) and if we want to read test data or configurations from XML.

`Javax.activation` and `javax.mail` are used for the Java mail API, and we will use them to write an email utility in Appendix A (Listing A-3).

`Mongo-java-driver` is required to host reports. The `poi` dependencies are used to work with Excel data. `Tess4J` is the tesseract for Java and is used in the OCR Util (Appendix A). `Itext` is used in PDF Util (Chapter 12) to create PDF reports, and `unirest` is used in QC integration (Chapter 18).

`Io.github.bonigarcia` is required to use `WebDriverManager` with tests without having to set specific JVM properties for each driver. We will use `io.appium:java-client` annotations to locate the page elements in each page object.

You will see that some dependencies are marked `implementation`, while others are marked as `testimplementation` or `compile` type. You can use these interchangeably, while `implementation` is a super-set of all.

Now, let us define some tasks and parallel execution settings in `build.gradle` just below the preceding snippet, as shown in Listing 3-2.

Listing 3-2. Build.gradle with Tasks and Parallel Execution Settings

```
task Smoke(type: Test, description: 'Run Smoke tests on App...') {
   useJUnitPlatform() { includeTags 'smoke' }
}

task Regression(type: Test, description: 'Run Regression tests
on App...') {
   useJUnitPlatform() { includeTags 'regression' }
}

task SIT(type: Test, description: 'Run SIT tests on App...') {
   useJUnitPlatform() { includeTags 'sittest' }
}

task AT(type: Test, description: 'Run AT tests on App...') {
   useJUnitPlatform() { includeTags 'attest' }
}

test {
   systemProperty "cucumber.options",
   System.getProperty("cucumber.options")
}

tasks.withType(Test){
   doFirst{
      systemProperties System.getProperties()
      systemProperties['junit.jupiter.execution.parallel.
      enabled'] = true
      systemProperties['junit.jupiter.execution.parallel.mode.
      default'] = "same_thread"
      systemProperties['junit.jupiter.execution.parallel.
      config.strategy'] = "fixed"
      systemProperties['junit.jupiter.execution.parallel.
      config.fixed.parallelism'] = 5
```

```
        systemProperties['junit.jupiter.execution.parallel.mode.
        classes.default'] = "concurrent"
    }
}
```

Tasks are defined as annotations in Gradle, so we need to create the annotations. This is why we created the `Annotations` folder.

Preparing gradle.properties

You will notice that for several dependencies, the version number is parameterized and is in curly brackets {}. These parameters are assigned values in the `gradle.properties` file. Right-click on your project root `testautomation` (Figure 3-2) ➤ New ➤ File ➤ enter file name as `gradle.properties`. Now, let's insert the parameter values, as in Listing 3-3.

Listing 3-3. gradle.properties

```
version=1.0-SNAPSHOT
gson_version=2.8.6
spring_boot_version=2.2.0.RELEASE
aspectj_version=1.8.10
report_version=1.0.12
allure_version=2.13.0
allure_junit_version=2.13.0
appium_version=7.3.0
junit_jupiter_engine_version=5.5.2
junit_jupiter_api_version=5.5.2
assertj_version=3.13.2
junit_jupiter_params_version=5.5.2
junit_jupiter_version=5.5.2
allurePluginVersion=2.13.5
```

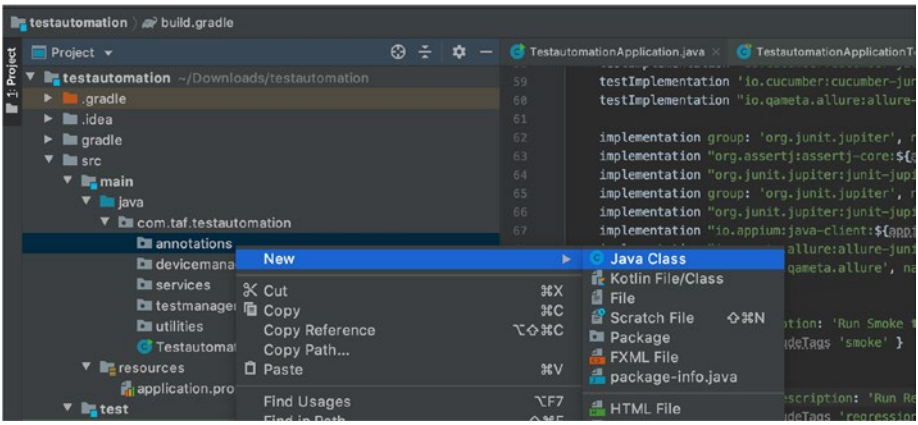


Figure 3-2. *Location of Gradle annotations*

Creating Annotations for Gradle Tasks

To create the annotations, right-click on the folder ➤ New ➤ Java Class ➤ Annotation, as shown in Figure 3-2.

Figure 3-3 shows the annotation for the AT task.

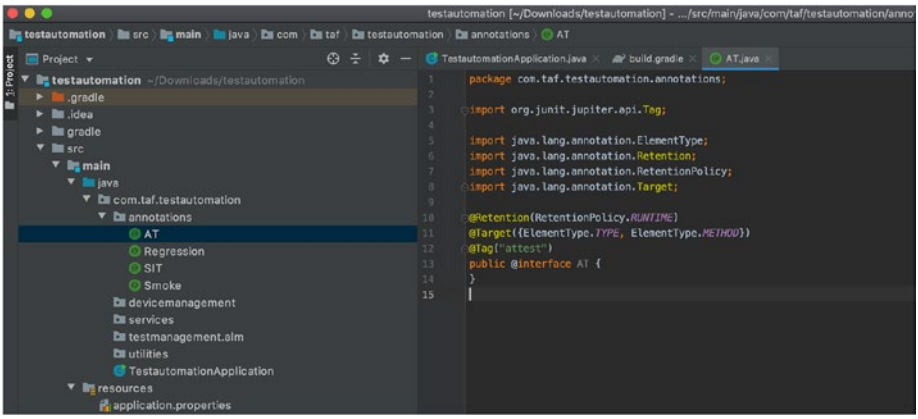


Figure 3-3. *Implementation for Gradle AT task*

In the same way, write the other three annotations. The only thing that changes other than the name is the tag at line 12.

Preparing settings.gradle

The `settings.gradle` file is used to specify build-level settings and is always executed before `build.gradle` is executed. In our case, the only build-level setting is the project root, as shown in Listing 3-4.

Listing 3-4. settings.gradle

```
rootProject.name = 'testautomation'
```

Summary

In this chapter, we configured Gradle for dependency management and task execution. Gradle automatically downloads the plugin libraries as you add them. We have explained the use case for each plugin and also explained how to define the Gradle tasks and associated annotations. In the next chapter, we will create properties files and learn various ways to read these files.

CHAPTER 4

Creating the Properties Files

In the previous chapter, we added Gradle plugins and dependencies to our project. We also defined the Gradle tasks for test execution. Now we are ready to start the serious coding required to build our framework. In this chapter, we will do that by creating a properties file that will be used to create driver objects in the following chapter. Let's begin.

Creating Your Properties Files

When we bootstrapped with Spring, we got one empty properties file. We can use this for API testing. For now, let's set up the property shown in Listing 4-1.

Listing 4-1. application.properties

```
prop.url=https://jsonplaceholder.typicode.com/users
```

Let's create another properties file for user interface (UI) testing and call it `uitest.properties`, as shown in Listing 4-2. Right-click on resources (src/main) ➤ New ➤ File ➤ uitest.properties and add the following entries, which I will explain shortly.

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_4

Listing 4-2. uitest.properties

```
#Flag for Json util
url=https://jsonplaceholder.typicode.com/users

#App account details and environment
username=testuser@gmail.com
password=Password1
userFirstName=Test
userLastName=User
environment=env

#Parameters required to create and select a specific driver
executionTarget=LOCAL
isMobile=true
isAndroid=true
isIos=false
browserName=
hubAddress=http://127.0.0.1:4723/wd/hub

#Flag to specify whether to load excel test data
loadExcel=Y

#Flag to specify timezone
timeZone=PST

#Flags to specify all the languages for localization testing
deviceLanguage=de-DE,en-GB,en-US,es-ES,fr-FR
language=en

#Pdf report
reportPrefix=./
mergedReport=./test-result/pdfreport/Report.pdf

#Excel data table location
```

```
dataTable1=src/test/resources/testdata/excel/testData1.xlsx
dataTable2=src/test/resources/testdata/excel/testData2.xlsx

#Data Sources
dataSource1=source1
dataSource2=source2

#build number
build=xxxx

#Android Desired Capabilites
platformName=android
platformVersion=10.0.0
deviceName=emulator-5554
noReset=true
automationName=UiAutomator2
appPackage=com.xxxx
appActivity=com.xxxx
app=/Users/{userId}/Downloads/xxxx.apk
appOld=/Users/{userId}/Downloads/xxxx.apk
defaultService=no

#iOS Desired Capabilities
iosPlatformName=iOS
iosPlatformVersion=13.6
iosUdid=xxxx
iosDeviceName=Koushik's iPhone
iosAutomationName=XCUITest
iosXcodeOrgId=xxxx
iosXcodeSigningId=iPhone Developer
iosApp=/Users/{userId}/Downloads/xxxx.ipa
iosBundleId=com.xxx
iosDefaultService=yes
```

```
iosNoReset=true
localization=no
appLanguage=en-US
showIOSLog=true

#See Test-related properties
useSeeTestClient=no
seeTestWdHub=
accessKey=
testNameIos=
testNameAndroid=
deviceQueryIos=@os='Ios' and starts-with(@name,'iPhone')
deviceQueryAndroid=@os='android' and @category='PHONE' and
@version=10.0'
appVersionIos=
appVersionAndroid=
newCommandTimeout=240
cloudAppIos=cloud:{appPackage}
cloudAppAndroid=cloud:{appPackage}/{appActivity}
instrumentApp=true

#ALM-related properties
almUsername=
almPassword=
defectID=
testRunID=
updateURL=https://xxxx/qcbin/rest/domains/xxxx/projects/
xxxx/runs/
deleteURL=https://xxxx/qcbin/rest/domains/xxxx/projects/
xxxx/runs/
updateKeyPass=status
updateValuePass=Passed
```

```
updateKeyFail=status
updateValueFail=Failed
name=TestScreen
id=xx
test-id=
test-name=
has-linkage=N
path=
cycle-id=
vc-version-number=
draft=N
status=Not Completed
owner=
test-config-id=
ver-stamp=
testcycl-name=
vc-locked-by=
duration=
testcycl-id=
subtype-id=hp.qc.run.MANUAL
attachment=Y
test-instance=1
cycle-name=
vc-status=
user-01=
user-02=
user-03=
user-04=
user-05=
user-06=
user-07=
```

```
user-08=
user-09=
user-10=
imageFilePath=/Users/{userId}/Documents/QCAttachent/
samplejpeg.jpeg
imageContentType=image/jpeg
docFilePath=/Users/{userId}/Documents/QCAttachent/
sampledoc.docx
docContentType=application/vnd.openxmlformats-officedocument.
wordprocessingml.document
pdfFilePath=/Users/{userId}/Documents/test-automation/
appautomation/test-result/pdfreport/Report.pdf
pdfContentType=application/pdf
excelFilePath=/Users/{userId}/Documents/QCAttachent/
sampexcel.xlsx
excelContentType=application/vnd.openxmlformats-officedocument.
spreadsheetml.sheet
textFilePath=/Users/{userId}/Documents/QCAttachent/
sampletext.txt
textContentType=text/plain
```

We will use the URL for API testing in Chapter 16. The next set of properties are for logging in to the application. This will change based on your specific application. The third set of parameters are required to create drivers based on the operating system in use. `isAndroid` would be true for Android and false for iOS. `isIos` would be true for iOS and false for Android. This framework can support web drivers also, though that is outside the scope of this book. Those of you interested can try this using `isMobile` and `browserName` parameters. However, tweaking the framework to support `WebElements` at the page-object level is best discussed in a new book.

The next two parameter sets are self-explanatory. We will discuss localization testing in Chapter 19, using the five languages we defined in the properties file. We will use PDF report parameters in PdfUtil class in Chapter 12. The location of Excel test data is defined in `dataTable1` and `dataTable2`. Data sources are defined in `dataSource1` and `dataSource2`, while the build number of the app is defined in the `build` parameter.

The next two sets of parameters are for creating Android and iOS drivers with the given desired capabilities. We will discuss this in Chapter 5. For now, I would like to mention that `appPackage`, `appActivity`, `iosBundleId`, `app`, `iosApp`, `iosUdid`, and `iosXcodeOrgId` will be specific to your app and devices. You can find `iosUdid` in several ways:

i. running the following command:
`instruments -s devices`

ii. in iTunes

iii. from XCode ➤ Window ➤ Devices and Simulators

iv. by running the following command (in case of real device): `ios-deploy -c`

I have also added the standard capabilities required if you want to use the `seeTest` client instead. The final set of parameters are required for HP QC connection. I will save the discussion of these parameters for Chapter 18.

You will notice that I have used `xxxx` as attribute values. This is because it will change based on the application you are using and its deployment. Henceforth, I will use `xxxx` or `yyyy` or `zzzz` in all such situations. `{userId}` would be replaced by your own user ID.

Now that we have created property files, in the next two sections we will discuss two different ways to read from them.

Reading from Properties File with Spring-Boot Library

Spring-Boot has a powerful set of annotations with which to read from multiple property file sources at once and capture them in POJO. Let's see how.

Create a new Java class named `TestAutomationProperties` in `src/main/java/com/taf/testautomation`, as shown in Figure 4-1.

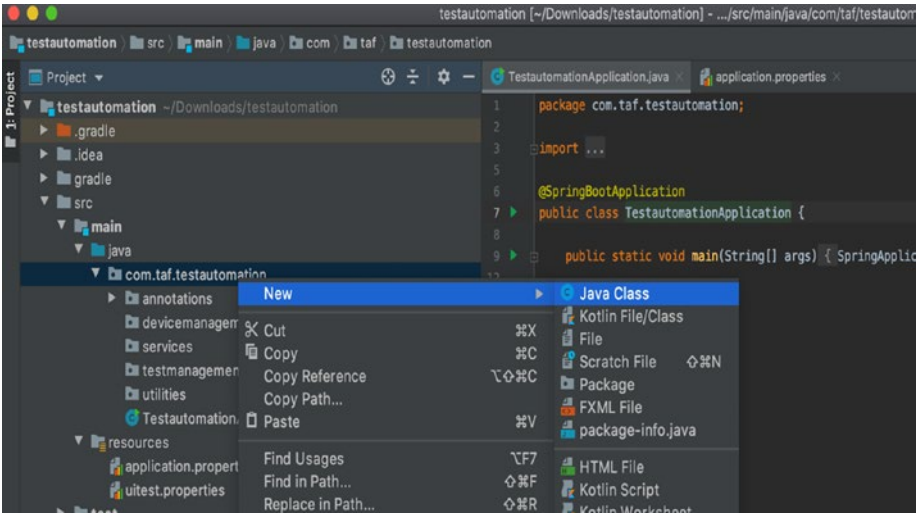


Figure 4-1. *TestAutomationProperties class location*

Put the code shown in Listing 4-3 in the `TestAutomationProperties` class.

Listing 4-3. TestAutomationProperties.java

```
package com.taf.testautomation;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```
import org.springframework.boot.context.properties.
ConfigurationProperties;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.context.annotation.PropertySources;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Configuration
@PropertySources( {
        @PropertySource("classpath:application.properties")
})

@ConfigurationProperties(prefix="prop")
public class TestAutomationProperties {
    private String url;
}
```

`@PropertySources` and `@PropertySource` are Spring annotations. Using these annotations, we can read from any property file. The property file path has to be mentioned as an argument in the `@PropertySource` annotation. We can have multiple property files—all we have to do is use multiple `@PropertySource` annotations with same `@PropertySources` annotation.

The `@Configuration` annotation is another Spring annotation. Any class with this annotation will be configured as a bean in the Spring context. Once configured as a bean, we can autowire the object of this class as a dependency injection anywhere else. A Spring container manages the dependency for this bean object. We will see this in API testing (Chapter 16).

You can write as many parameters as you want in `application.properties`. Just make sure you are writing the corresponding variables in the `TestAutomationProperties` class. You will see that I have not written any getter/setter methods in the `TestAutomationProperties` class. Welcome to Lombok! The `@Data` annotation in Lombok takes care of that while reducing boilerplate code. `@AllArgsConstructor` and `@NoArgsConstructor` are the other two Lombok annotations here. These do away with the need to write constructors.

Finally, the `@ConfigurationProperties` Spring annotation lets you specify prefixes used in properties files. In our case, the prefix is `prop`, so the prefix would be ignored during variable assignment while reading from the properties file.

In the next section, we will discuss other ways of reading from properties files.

Reading from Properties Files in the Traditional Ways

The traditional ways of reading a properties file are either (i) using `java.util.Properties` class methods or (ii) by reading line by line from the properties file using the `readLine()` method of the `BufferedReader` class. We will see these in Chapters 5 and 9.

Summary

In this chapter, we created properties files and learned the ways to read from them. You are now all set to begin your journey into the world of Appium! In the next chapter, we will create Appium sessions and driver objects by reading from these properties files.

CHAPTER 5

Creating Android, iOS, and Web Drivers on Demand

In the previous chapter, we created properties files and learned how to read them using the Spring-Boot library, as well as in the traditional way. In this chapter, we will learn how to create a driver instance with the desired set of capabilities in the property file. Let's dive in!

Creating a Driver with Standard Desired Capabilities

Let's create a class, `Session`, in `src/test/java/com/taf/testautomation`, as shown in Listing 5-1.

Listing 5-1. Session.java

```
package com.taf.testautomation;

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


K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_5

```
import io.appium.java_client.ios.IOSDriver;
import io.appium.java_client.remote.
AndroidMobileCapabilityType;
import io.appium.java_client.remote.MobileCapabilityType;
import io.appium.java_client.service.local.
AppiumDriverLocalService;
import io.appium.java_client.service.local.
AppiumServiceBuilder;
import lombok.Setter;
import lombok.extern.slf4j.Slf4j;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.firefox.FirefoxOptions;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;

import java.io.*;
import java.net.URL;
import java.util.HashMap;
import java.util.Properties;
import java.util.concurrent.TimeUnit;

@Setter
@Slf4j
public class Session {

    private WebDriver webDriver;
    private AppiumDriver<MobileElement> appiumDriver;
    private AndroidDriver<MobileElement> androidDriver;
    private IOSDriver<MobileElement> iosDriver;
    private Boolean isMobile;
    private HashMap<String, String> customProperties = new
    HashMap<>();
```

```
protected String prevBuild = "no";

Reader reader;
ClassLoader loader = this.getClass().getClassLoader();
URL myURL = loader.getResource("uitest.properties");
String path = myURL.getPath();

{
    try {
        reader = new FileReader(path);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    Properties prop = new Properties();
    try {
        prop.load(reader);
    } catch (IOException e) {
        e.printStackTrace();
    }
    prop.forEach((k, v) -> customProperties.put(k.
    toString(), v.toString()));
}

public void startSession() throws Exception {
    if (customProperties.get("executionTarget").
    isEmpty()) {
        throw new RuntimeException("Session configuration
        error, execution target is empty");
    } else if (customProperties.get("executionTarget").
    equals("LOCAL")) {
        startLocalSession();
    } else if (customProperties.get("executionTarget").
    equals("GRID")) {
```

```
            startGridSession();
        } else if (customProperties.get("executionTarget").
        equals("EXPERITEST")) {
            startExperiTestSession();
        }
    }

    private void startLocalSession() throws IOException {
        StringBuilder urlString = new StringBuilder();
        urlString.append(customProperties.get("hubAddress"));
        URL url = new URL(urlString.toString());
        DesiredCapabilities desiredCapabilities = this.
        getDesiredCapabilities();
        this.startRemoteSession(url, desiredCapabilities);
    }

    private void startRemoteSession(URL url, final
    DesiredCapabilities desiredCapabilities) throws IOException {
        if (Boolean.parseBoolean(customProperties.
        get("isMobile"))) {
            if (Boolean.parseBoolean(customProperties.
            get("isAndroid"))) {
                if (customProperties.get("defaultService").
                equals("no")) {
                    this.androidDriver = new AndroidDriver(url,
                    desiredCapabilities);
                    this.appiumDriver = this.androidDriver;
                } else {
                    Runtime.getRuntime().exec("./command.txt");
                    AppiumDriverLocalService service = Appium
                    DriverLocalService.buildDefaultService();
                    service.start();
```

```
                this.androidDriver = new
                AndroidDriver(service, desiredCapabilities);
                 this.appiumDriver = this.androidDriver;
            }
        } else if (Boolean.parseBoolean(customProperties.
        get("isIos"))) {
            if (customProperties.get("iosDefaultService").
            equals("no")) {
                this.iosDriver = new IOSDriver(url,
                desiredCapabilities);
                this.appiumDriver = this.iosDriver;
            } else {
                File testLogFile = new File("./log.txt");
                Runtime.getRuntime().exec("./command.txt");
                AppiumDriverLocalService service
                = new AppiumServiceBuilder().
                withLogFile(testLogFile).build();
                service.start();
                this.iosDriver = new IOSDriver(service,
                desiredCapabilities);
                this.appiumDriver = this.iosDriver;
            }
            this.webDriver = this.appiumDriver;
        } else {
            RemoteWebDriver remoteWebDriver = new
            RemoteWebDriver(url, desiredCapabilities);
            this.webDriver = remoteWebDriver;
            this.webDriver.manage().window().maximize();
        }
    }
}
```

```
    private void startGridSession() {

    }

    private void startExperiTestSession() throws IOException {
        StringBuilder urlString = new StringBuilder();
        urlString.append(customProperties.get("seeTestWdHub"));
        URL url = new URL(urlString.toString());
        DesiredCapabilities desiredCapabilities = this.
        getDesiredCapabilitiesSeeTest();
        this.startRemoteSession(url, desiredCapabilities);
    }

    public DesiredCapabilities getDesiredCapabilities() {
        DesiredCapabilities desiredCapabilities = new
        DesiredCapabilities();
        if (Boolean.parseBoolean(customProperties.
        get("isAndroid"))) {
            desiredCapabilities.setCapability("platformName",
            customProperties.get("platformName"));
            desiredCapabilities.setCapability("platform
            Version", customProperties.get("platformVersion"));
            desiredCapabilities.setCapability("deviceName",
            customProperties.get("deviceName"));
            desiredCapabilities.setCapability("noReset",
            customProperties.get("noReset"));
            desiredCapabilities.setCapability("appPackage",
            customProperties.get("appPackage"));
            desiredCapabilities.setCapability("appActivity",
            customProperties.get("appActivity"));
            desiredCapabilities.setCapability("automationName",
            customProperties.get("automationName"));
            if (prevBuild.equals("yes")) {
```

```
            desiredCapabilities.setCapability("app",
            customProperties.get("appOld"));
        } else {
            desiredCapabilities.setCapability("app",
            customProperties.get("app"));
        }
    }
    if (Boolean.parseBoolean(customProperties.
    get("isIos"))) {
        if (customProperties.get("localization").
        equals("yes")) {
            System.out.println("localization value " +
            customProperties.get("localization"));
            System.out.println("language value " +
            customProperties.get("appLanguage"));
            desiredCapabilities.
            setCapability("platformName", customProperties.
            get("iosPlatformName"));
            desiredCapabilities.setCapability("platformVersi
            on", customProperties.get("iosPlatformVersion"));
            desiredCapabilities.setCapability("deviceName",
            customProperties.get("iosDeviceName"));
            desiredCapabilities.setCapability("app",
            customProperties.get("iosApp"));
            desiredCapabilities.setCapability("noReset",
            customProperties.get("iosNoReset"));
            desiredCapabilities.
            setCapability("automationName",
            customProperties.get("iosAutomationName"));
            desiredCapabilities.setCapability("udid",
            customProperties.get("iosUdid"));
```

```
            desiredCapabilities.setCapability("xcodeOrgId",
            customProperties.get("iosXcodeOrgId"));
            desiredCapabilities.
            setCapability("xcodeSigningId",
            customProperties.get("iosXcodeSigningId"));
            desiredCapabilities.setCapability("language",
            customProperties.get("appLanguage"));
            desiredCapabilities.setCapability("locale",
            customProperties.get("appLanguage"));
        } else {
            desiredCapabilities.
            setCapability("platformName", customProperties.
            get("iosPlatformName"));
            desiredCapabilities.setCapability
            ("platformVersion", customProperties.
            get("iosPlatformVersion"));
            desiredCapabilities.setCapability("deviceName",
            customProperties.get("iosDeviceName"));
            desiredCapabilities.setCapability("app",
            customProperties.get("iosApp"));
            desiredCapabilities.setCapability("noReset",
            customProperties.get("iosNoReset"));
            desiredCapabilities.
            setCapability("automationName",
            customProperties.get("iosAutomationName"));
            desiredCapabilities.setCapability("udid",
            customProperties.get("iosUdid"));
            desiredCapabilities.setCapability("xcodeOrgId",
            customProperties.get("iosXcodeOrgId"));
            desiredCapabilities.
            setCapability("xcodeSigningId",
            customProperties.get("iosXcodeSigningId"));
```

```
            desiredCapabilities.setCapability("showIOSLog",
            customProperties.get("showIOSLog"));
        }
    }
    if (customProperties.get("browserName").
    equals("chrome")) {
        ChromeOptions chromeOptions = getChromeOptions();
        desiredCapabilities.setCapability(ChromeOptions.CAP
        ABILITY, chromeOptions);
    }
    if (customProperties.get("browserName").
    equals("firefox")) {
        FirefoxOptions firefoxOptions =
        getFireFoxOptions();
        desiredCapabilities.setCapability(ChromeOptions.CAP
        ABILITY, firefoxOptions);
    }
    return desiredCapabilities;
}

public DesiredCapabilities
getDesiredCapabilitiesSeeTest() {
    DesiredCapabilities desiredCapabilities = new
    DesiredCapabilities();
    if (Boolean.parseBoolean(customProperties.
    get("isAndroid"))) {
        desiredCapabilities.setCapability("accessKey",
        customProperties.get("accessKey"));
        desiredCapabilities.setCapability("testName",
        customProperties.get("testNameAndroid"));
        desiredCapabilities.setCapability("deviceQuery",
        customProperties.get("deviceQueryAndroid"));
```

```
            desiredCapabilities.setCapability(MobileCapabilityT
            ype.APP, customProperties.get("cloudAppAndroid"));
            desiredCapabilities.setCapability(AndroidMobil
            eCapabilityType.APP_PACKAGE, customProperties.
            get("appPackage"));
            desiredCapabilities.setCapability(AndroidMobileCapa
            bilityType.APP_ACTIVITY, customProperties.get
            ("appActivity"));
        }
        if (Boolean.parseBoolean(customProperties.get("isIos"))) {
            desiredCapabilities.setCapability("accessKey",
            customProperties.get("accessKey"));
            desiredCapabilities.setCapability("testName",
            customProperties.get("testNameIos"));
            desiredCapabilities.setCapability("deviceQuery",
            customProperties.get("deviceQueryIos"));
            desiredCapabilities.setCapability("appVersion",
            customProperties.get("appVersionIos"));
            desiredCapabilities.setCapability("platformName",
            customProperties.get("iosPlatformName"));
           desiredCapabilities.setCapability("newCommandTimeout",
           customProperties.get("newCommandTimeout"));
            desiredCapabilities.setCapability("automationName",
            customProperties.get("iosAutomationName"));
            desiredCapabilities.setCapability("app",
            customProperties.get("cloudAppIos"));
            desiredCapabilities.setCapability("bundleId",
            customProperties.get("iosBundleId"));
        }
        return desiredCapabilities;
    }
```

```
public Boolean isMobile() {
    return isMobile;
}

private void setMobile(Boolean mobile) {
    isMobile = mobile;
}

public AppiumDriver<MobileElement> getAppiumDriver() {
    if (Boolean.parseBoolean(customProperties.
    get("isMobile"))) {
        return this.appiumDriver;
    }
    throw new IllegalArgumentException("Appium driver
    requested, but session is not configured to use an
    Appium driver");
}

public AndroidDriver<MobileElement> getAndroidDriver() {
    if (Boolean.parseBoolean(customProperties.
    get("isAndroid"))) {
        return this.androidDriver;
    }
    throw new IllegalArgumentException("Android driver
    requested, but session is not configured to use an
    Android driver");
}

public IOSDriver<MobileElement> getIosDriver() {
    if (Boolean.parseBoolean(customProperties.
    get("isIos"))) {
        return this.iosDriver;
    }
```

```
            throw new IllegalArgumentException("IOS driver
            requested, but session is not configured to use an iOS
            driver");
        }

        public WebDriver getWebDriver() {
            this.setMobile(Boolean.parseBoolean(customProperties.
            get("isMobile")));
            if (!isMobile()) {
                getWebDriver().manage().timeouts().
                pageLoadTimeout(30L, TimeUnit.SECONDS);
            }
            return this.webDriver;
        }

        protected ChromeOptions getChromeOptions() {
            ChromeOptions chromeOptions = new ChromeOptions();
            return chromeOptions;
        }

        protected FirefoxOptions getFireFoxOptions() {
            FirefoxOptions firefoxOptions = new FirefoxOptions();
            return firefoxOptions;
        }

        public HashMap<String, String> getCustomProperties() {
            return customProperties;
        }

        public void closeSession() {
            if (Boolean.parseBoolean(customProperties.
            get("isMobile"))) {
                try {
```

```
                if (customProperties.get("isAndroid").
                equals("true")) {
                    appiumDriver.resetApp();
                } else if (!appiumDriver.
                removeApp(customProperties.get("iosBundleId"))) {
                    appiumDriver.resetApp();
                }
            } catch (Exception e) {
                log.error("Error closing App");
                e.printStackTrace();
            }
            try {
                appiumDriver.quit();
            } catch (Exception e) {
                log.error("Error closing session");
                e.printStackTrace();
            }
        } else {
            getWebDriver().quit();
        }
    }
}
```

The non-static block at the beginning of the `Session` class loads the `uitests.properties` in the map named `customProperties`. I use a non-static block as I pass the same session object to multiple page object constructors, thereby obviating the need for a static block.

I use three separate methods—`startLocalSession()`, `startGridSession()`, and `startExperiTestSession()`—to embed the logic of creating drivers in the corresponding setups. Each of these methods creates the corresponding Appium server URL and desired capabilities before calling the `startRemoteSession` method to create

the corresponding driver. The URLs for the servers are obtained from the `uitest.properties` file. In this book, I have implemented `startLocalSession()` and `startExperiTestSession()`. I leave the `startGridSession()` method as an exercise for you.

The method `getDesiredCapabilities()` embeds the logic to create the desired capabilities for each driver in each setup. You will see four high-level `if` loops for Android, iOS, Chrome, and Firefox. The last two are outside the scope of this book.

In the `if` loop for Android, I use one protected variable called `prevBuild` so that I can just call `session.setPrevBuild("yes")` from my test suite (the Lombok `@Setter` annotation means we do not need to write setter methods), where session is the `Session` object. We will see this in Chapter 14, where we will discuss how to test multiple versions of the same app.

In the `if` loop for iOS, I use the `localization` parameter for localization testing (Chapter 19). When localization is set to yes, I use two additional desired capabilities named `language` and `locale` to launch the app in the desired language. The capability `showIOSLog` was used to print the device log along with the Appium log in the console. This is useful for iOS. In Android, we can use `adb` to print the device log.

Once the Android or iOS driver is created, it can be referenced by using the single object reference `appiumDriver`. I have written separate getter methods for these drivers.

Now, let's look at another way of creating an Appium service.

Creating a Driver with Default Service

With the `startRemoteSession` method, you can see that I use the property `defaultService` to determine if I will use the `AppiumDriverLocalService` methods to create the Appium server or use the server URL in `uitest.properties`. For iOS, I have set the parameter `iosDefaultService`

to yes to show you both ways to use Appium server. Since I'm in iOS, I have used the `showIOSLog` flag, and I create the log file before calling the `AppiumDriverLocalService` methods. You can use either `AppiumDriverLocalService.buildDefaultService()` or the new `AppiumServiceBuilder().build()` to create `AppiumDriverLocalService` objects. Then, just use the `start()` method to start the Appium server. You can also see I run the script `command.txt` before creating the default service so as to terminate any Appium session already running on the computer:

```
Runtime.getRuntime().exec("./command.txt");
```

Listing 5-2 shows the `command.txt` script file. To create the file, right-click on the project root `testautomation` ➤ New ➤ File ➤ enter the full file name `command.txt`. In Chapter 15, we will discuss running scripts in more detail.

Listing 5-2. command.txt

```
#! /bin/bash
cd ~
lsof -t -i tcp:4723 | xargs kill -9
```

Please note I am using a Mac. The preceding shell script would work in Linux also. In Windows, you have to create a text file like Listing 5-3 and save it with .bat extension.

Listing 5-3. command.bat

```
@ECHO OFF
FOR /F "tokens=5 delims= " %%P IN ('netstat -a -n -o ^| findstr
:8080') DO TaskKill.exe /PID %%P
PAUSE
```

Creating Drivers for Grid or Cloud Execution

I have created a separate method for the SeeTest client to create the desired capabilities, so as to not confuse you. I encourage those of you with SeeTest access to try this as well.

Quitting Driver and Teardown

Finally, the `closeSession()` method does the teardown process. The `teardown()` process is required at the end of every test suite (details in Chapter 8) in order to start with a clean slate for the next execution. In the case of Android, I reset the app, whereas in iOS, I remove the app (first `try` block) before calling the `quit()` method (second `try` block) on the driver. This is because, for Android, the reset method works reasonably well and there is no need to remove the app. However, you can use the logic that works best for you.

Summary

In this chapter, we learned how to create Android and iOS drivers with the desired capability flags set in the property file. We also learned how to implement `teardown()`. In the next chapter, we will create a class to capture all common mobile actions like tap, press, click, swipe, checking for existence of elements, etc. that would be extended by all page objects.

CHAPTER 6

Enhancing the Framework: Common Mobile Actions

In the previous chapter, we learned how to create Android and iOS drivers based on flags in the `uitest.properties` file. In this chapter, we will create a class to implement common page object actions, like wait, click, type, swipe, scroll, press, and so on, and to define variables that will be used in all page objects. This class will be extended by all page objects. Let's begin.

Creating Variables for the MobileBaseActionScreen Class

Let's create two enums, `Direction` and `SwipeDirection`, in the `com/taf/testautomation` folder (see screenshot in Figure 8-1). I use two different enums to show the different ways in which we can write them. I use the first, as shown in Listing 6-1, for scroll actions, and the second, as shown in Listing 6-2, for swipe actions.

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_6

Listing 6-1. Direction Enum

```
package com.taf.testautomation;

public enum Direction
{
    UP,
    DOWN,
    LEFT,
    RIGHT;
}
```

Listing 6-2. SwipeDirection enum

```
package com.taf.testautomation;

public enum SwipeDirection {
    RIGHT("right"),
    LEFT("left");

    private String direction;

    SwipeDirection(String direction) {
        this.direction = direction;
    }

    public String direction() {
        return direction;
    }

}
```

Let's also create a class `PdfUtil` within the `src/main/java/com/taf/testautomation/utilities/pdfutil` folder and include an empty static method, as shown in Listing 6-3, so as to avoid getting an error in the `MobileBaseActionScreen` that we will create next. We will use `PdfUtil` in Chapter 12, so I will save the details till then to avoid complexity.

Listing 6-3. Empty Method in PdfUtil Class

```
public static void getPdfFromImage(String image, String
pdfFile) { }
```

Now, let's create the class `MobileBaseActionScreen`, as shown in Listing 6-4, within the `screens` folder (screenshot in Figure 7-1). In this section and in the next two sections I will explain the code.

Listing 6-4. MobileBaseActionScreen.java

```
package com.taf.testautomation.screens;

import com.taf.testautomation.Direction;
import com.taf.testautomation.SwipeDirection;
import com.taf.testautomation.utilities.pdfutil.PdfUtil;
import io.appium.java_client.MobileDriver;
import com.taf.testautomation.Session;
import io.appium.java_client.MobileElement;
import io.appium.java_client.TouchAction;
import io.appium.java_client.android.Activity;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;
import io.appium.java_client.touch.WaitOptions;
import io.appium.java_client.touch.offset.PointOption;
import io.qameta.allure.Step;
import lombok.extern.slf4j.Slf4j;
import org.assertj.core.api.SoftAssertions;
import org.openqa.selenium.*;
import org.openqa.selenium.support.ui.FluentWait;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
```

```
import java.io.IOException;
import java.time.Duration;
import java.util.HashMap;
import java.util.concurrent.TimeUnit;
import java.util.stream.IntStream;

import static io.appium.java_client.MobileBy.
AndroidUIAutomator;
import static io.appium.java_client.touch.WaitOptions.
waitOptions;
import static io.appium.java_client.touch.offset.
PointOption.point;
import static java.lang.String.format;
import static java.time.Duration.ofMillis;
import static org.openqa.selenium.support.
ui.ExpectedConditions.*;

@Slf4j
public class MobileBaseActionScreen {

    private Session session;
    public static final int MIN_WAIT = 2;
    public static final int SMALL_WAIT = 10;
    public static final int MED_WAIT = 30;
    public static final int LONG_WAIT = 60;
    public static final int MAX_WAIT = 120;
    public static final int MAX_SCROLL = 100;
    protected static String email;
    protected static String password;
    protected static String firstName;
    protected static String lastName;

    public MobileBaseActionScreen(Session session) {
```

```
    this.session = session;
}

public Session getSession() {
    return this.session;
}

public SoftAssertions getSoftAssertions() {
    if (null != this.getSession()) {
        return new SoftAssertions();
    }
    throw new IllegalArgumentException("Please initialize
    session before attempting to access Soft Assertion");
}

/**
 * Checking that an element is present on the screen
 and visible.
 *
 * @param element
 * @param timeout
 * @return True if the element is visible otherwise
 return false.
 */
protected boolean doesElementExist(MobileElement element,
int timeout) {
    try {
        waitOn(getSession().getAppiumDriver(), timeout).
        until(visibilityOf(element));
    } catch (Exception toe) {
        return false;
    }
```

```
        return true;
    }

    /**
     * An implementation of the Wait interface with its
     timeout and
     * polling interval configured on the fly.
     *
     * @param <K>
     * @param object
     * @param timeOutSeconds
     * @return the fluent wait
     */
    protected <K> FluentWait<K> waitOn(K object, int
    timeOutSeconds) {
        return new FluentWait<>(object).
        ignoring(NoSuchElementException.class)
                .ignoring(NoSuchElementException.class)
                .ignoring(StaleElementReferenceException.class)
                .ignoring(Exception.class)
                .withTimeout(Duration.
                ofSeconds(timeOutSeconds));
    }

    /**
     * Gets the waitOn instance.
     *
     * @return A FluentWait instance.
     */
    public FluentWait<MobileDriver<MobileElement>> getWait() {
        return waitOn(getSession().getAppiumDriver(),
        MAX_WAIT);
    }
```

```
/**
 * Retrieves a TouchAction instance.
 *
 * @return a TouchAction instance.
 */
protected TouchAction getTouchAction() {
    return new TouchAction(getSession().getAppiumDriver());
}

/**
 * This method performs touch action based
 * on the coordinates provided
 *
 * @param width
 * @param start
 * @param end
 */
public void touchAction(int width, int start, int end) {
    getTouchAction().press(point(width, start))
            .waitAction(waitOptions(ofMillis(1000)))
            .moveTo(point(width, end))
            .release()
            .perform();
}

/**
 * Swipe action by touch between two points
 *
 * @param startx
 * @param starty
 * @param endx
```

```
     * @param endy
     * @param duration in milliseconds
     */
    protected void swipeTouchAction(int startx, int starty, int
    endx, int endy, int duration) {
        getTouchAction().press(PointOption.point(startx, starty))
                .waitAction(WaitOptions.waitOptions(Duration.
                ofMillis(duration)))
                .moveTo(PointOption.point(endx, endy))
                .release()
                .perform();
    }

    /**
     * Tap on.
     *
     * @param element the MobileElement
     */
    protected void tapOn(MobileElement element) {
        getTouchAction().tap(point(element.getLocation().
        getX(), element.getLocation().getY())).perform();
    }

    /**
     * Tap on element by co-ordinatess
     *
     * @param x co-ordinate,
     * @param y co-ordinate
     */
    protected void tapByCoordinates(int x, int y) {
        getTouchAction().tap(point(x, y)).perform();
    }
```

```
/**
 * Wait in seconds.
 *
 * @param seconds
 */
protected void waitInSeconds(int seconds) {
    try {
        TimeUnit.SECONDS.sleep(seconds);
    } catch (InterruptedException e) {
        log.error("Interrupted!", e);
        Thread.currentThread().interrupt();
    }
}

/**
 * Wait for displayed element.
 *
 * @param element
 * @param seconds
 */
protected void waitForDisplayedElement(MobileElement
element, int seconds) {
    waitOn(getSession().getAppiumDriver(), seconds).until(e
    lementToBeClickable(element));
}

/**
 * Wait for a given element instance to be invisible.
 *
 * @param element
 * @param timeout
 */
```

```
    protected void waitForInvisibilityOf(MobileElement element,
    int timeout) {
        try {
            waitOn(getSession().getAppiumDriver(), timeout).
            until(invisibilityOf(element));
        } catch (Exception e) {
            log.info("The element is not displayed");
        }
    }

    /**
     * Get y coordinate of the center.
     *
     * @param element
     * @return value
     */
    public int getYCenter(MobileElement element) {
        int upperY = element.getLocation().getY();
        int lowerY = upperY + element.getSize().getHeight();
        return (upperY + lowerY) / 2;
    }

    /**
     * Get x coordinate of the center.
     *
     * @param element
     * @return the x center
     */
    public int getXCenter(MobileElement element) {
        int leftX = element.getLocation().getX();
        int rightX = leftX + element.getSize().getWidth();
        return (rightX + leftX) / 2;
    }
```

```
/**
 * Get the X location of an element.
 *
 * @param element
 * @return X value of element
 */
protected Integer getX(MobileElement element) {
    return element.getLocation().getX();
}

/**
 * Get the Y location of an element.
 *
 * @param element
 * @return Y value of element
 */
protected Integer getY(MobileElement element) {
    return element.getLocation().getY();
}

/**
 * Get the width size of an element.
 *
 * @param element
 * @return width value of element
 */
protected Integer getWidth(MobileElement element) {
    return element.getSize().getWidth();
}

/**
 * Get the Height size of an element.
 *
```

```
     * @param element
     * @return Height value of element
     */
    protected Integer getHeight(MobileElement element) {
        return element.getSize().getHeight();
    }

    /**
     * Get end position in x integer
     *
     * @param element
     * @return the integer
     */
    public Integer getEndPositionInX(MobileElement element) {
        return getX(element) + getWidth(element);
    }

    /**
     * Get end position in y integer
     *
     * @param element
     * @return the integer
     */
    public Integer getEndPositionInY(MobileElement element) {
        return getY(element) + getHeight(element);
    }

    /**
     * Compares Y axis position between two elements
     * Checks if element1 is below element2
     * @param element1
     * @param element2
```

```
 * @return True if first element is displayed below
 second element
 * else false
 */
public boolean checkIfFirstElementIsDisplayedBelowSecond
Element(MobileElement element1, MobileElement element2) {
    return element1.getLocation().getY() > element2.
    getLocation().getY();
}

/**
 * Compares X axis position between two elements
 * Checks if element2 is at right of element1
 * @param element1
 * @param element2
 * @return true, if element2 is at right side of element1
 */
public boolean isElementRightSideOfOther(MobileElement
element1, MobileElement element2) {
    return element1.getLocation().getX() < element2.
    getLocation().getX();
}

/**
 * Send app to background
 *
 * @param seconds
 */
@Step("Send app to background")
public void sendAppToBackground(int seconds) {
    log.info("Send app to background " + seconds + " seconds");
```

```
        getSession().getAppiumDriver().
        runAppInBackground(Duration.ofSeconds(seconds));
    }

    /**
     * Click on a given MobileElement element.
     *
     * @param element
     */
    protected void click(MobileElement element) {
        getWait().until(elementToBeClickable(element)).click();
    }

    /**
     * Use this method to simulate typing into an element,
     which may set its
     * value.
     *
     * @param element
     * @param text
     */
    public void type(MobileElement element, String text) {
        getWait().until(elementToBeClickable(element));
        element.clear();
        element.sendKeys(text);
    }

    /**
     * Use this method to simulate click and typing into an
     element, which may set its
     * value.
     *
     * @param element
```

```
 * @param text
 */
public void clickAndType(MobileElement element,
String text) {
    getWait().until(elementToBeClickable(element));
    element.click();
    element.clear();
    element.sendKeys(text);
}

/**
 * Use this method to swipe horizontally
 * @param startPoint the start X coordinate
 * @param endPoint the end X coordinate
 * @param anchor the Y coordinate
 */
private void swipeAction(int startPoint, int endPoint, int
anchor) {
    getTouchAction().press(point(startPoint, anchor))
            .waitAction(waitOptions(ofMillis(1000)))
            .moveTo(point(endPoint, anchor))
            .release()
            .perform();
  }

/**
 * Higher-level method for swipeActionDirectional
 * @param direction the horizontal swipe direction
 */
public void horizontalSwipe(SwipeDirection direction) {
    waitInSeconds(MIN_WAIT);
    Dimension size = getSession().getAppiumDriver().
    manage().window().getSize();
```

```
        int anchor = size.getHeight();
        int startPoint = size.getWidth();
        int endPoint = size.getWidth();
        swipeActionDirectional(direction, anchor, startPoint,
        endPoint, 0.50);
    }

    /**
     * Higher level method for swipeActionDirectional. Swipes
     end to end
     * @param direction the horizontal swipe direction
     */
    public void horizontalSwipeAcrossWidth(SwipeDirection
    direction) {
        waitInSeconds(MIN_WAIT);
        Dimension size = getSession().getAppiumDriver().
        manage().window().getSize();
        int startPoint = 0, endPoint = 0;
        int anchor = size.getHeight();
        if (direction.name()
                .equalsIgnoreCase(SwipeDirection.RIGHT.
                direction())) {
            startPoint = size.getWidth();
            endPoint = size.getWidth();
        } else {
            startPoint = 1;
            endPoint = size.getWidth();
        }

        swipeActionDirectional(direction, anchor, startPoint,
         endPoint, 0.50);
    }
```

```
/**
 * Horizontal swipe with element Y position as anchor.
 * @param element the MobileElement
 * @param direction the horizontal swipe direction
 */
public void horizontalSwipeOnContent(MobileElement element,
SwipeDirection direction) {
    scrollFastToElement(element);
    int yCenter = getYCenter(element);
    int startPoint = 0;
    int endPoint = 0;
    Dimension size = getSession().getAppiumDriver().
    manage()
            .window()
            .getSize();
    if (direction.equals(SwipeDirection.RIGHT)) {
        startPoint = getXCenter(element);
        endPoint = (int) (size.getWidth() * 0.1);
    } else {
        startPoint = getXCenter(element);
        endPoint = (int) (size.getWidth() * 0.9);
    }

    swipeTouchAction(startPoint, yCenter, endPoint,
     yCenter, 500);
 }

/**
 * This calls the swipeAction method for a given swipe
 direction.
 * @param direction the horizontal swipe direction
 * @param percentage used to update the anchor
 */
```

```
    private void swipeActionDirectional(SwipeDirection
    direction, int anchor, int startPoint, int endPoint, double
    percentage) {
        if (direction.name()
                .equalsIgnoreCase(SwipeDirection.RIGHT.
                direction())) {
            anchor = (int) (anchor * percentage);
            startPoint = (int) (startPoint * 0.8);
            endPoint = (int) (endPoint * 0.2);
        } else {
            anchor = (int) (anchor * percentage);
            startPoint = (int) (startPoint * 0.2);
            endPoint = (int) (endPoint * 0.8);
        }
        log.info("Swiping " + direction.name() + "...");
        swipeAction(startPoint, endPoint, anchor);
    }

    /**
     * This method is used to scroll vertically.
     * @param direction the vertical scroll direction
     */
    protected void scroll(Direction direction) {
        HashMap<String, String> scrollObject = new
        HashMap<String, String>();
        scrollObject.put("direction", direction.name().
        toLowerCase());
        getSession().getAppiumDriver().executeScript("mobile:
        scroll", new Object[]{scrollObject});
    }
```

```
 /**
  * This method calls scrollUpTouchAction for MAX_SCROLL.
  */
 public void scrollToTop() {
     int scrollCount = 1;
     while (scrollCount <= MAX_SCROLL) {
         scrollUpTouchAction();
         scrollCount++;
     }
}

 /**
  * This method calls scrollUpTouchAction.
  * @param count the initial count
  */
 public void scrollToTop(int count) {
     int scrollCount = count;
     while (scrollCount <= MAX_SCROLL) {
         scrollUpTouchAction();
         scrollCount++;
     }
 }

 /**
  * This method is used for scrolling up.
  */
 public void scrollUpTouchAction() {
     Dimension size = getSession().getAppiumDriver().
     manage().window().getSize();

     int width = size.getWidth() / 2;
     int start = (int) (size.getHeight() * 0.2);
     int end = (int) (size.getHeight() * 0.8);
```

```
        touchAction(width, start, end);
    }

    /**
     * This method is used for scrolling up, slowly.
     */
    public void slowScrollUpTouchAction() {
        Dimension size = getSession().getAppiumDriver().
        manage().window().getSize();
        int width = size.getWidth() / 2;
        int start = (int) (size.getHeight() * 0.6);
        int end = (int) (size.getHeight() * 0.8);
        touchAction(width, start, end);
     }

    /**
     * This method is used for scrolling down, faster.
     */
    public void fastScrollDownTouchAction() {
        Dimension size = getSession().getAppiumDriver().
        manage().window().getSize();
        int width = size.getWidth() / 2;
        int start = (int) (size.getHeight() * 0.8);
        int end = (int) (size.getHeight() * 0.2);
        touchAction(width, start, end);
    }

    /**
     * This method is used for scrolling down, slower.
     */
    public void slowScrollDownTouchAction() {
        Dimension size = getSession().getAppiumDriver().
        manage().window().getSize();
```

```
    int width = size.getWidth() / 2;
    int start = (int) (size.getHeight() * 0.8);
    int end = (int) (size.getHeight() * 0.6);
    touchAction(width, start, end);
}

/**
 * Scroll fast to bottom.
 *
 * @param element
 * @return the element
 */
public MobileElement scrollFastToElement(MobileElement
element) {

    for (int scroll = 0; scroll < MAX_SCROLL; scroll++) {
        try {
            if (element.isDisplayed())
                return element;
            else
                fastScrollDownTouchAction();
        } catch (Exception e) {
            fastScrollDownTouchAction();
        }
    }
    return null;
}

/**
 * Scroll fast to bottom.
 *
 * @param element
 */
```

```
    public void scrollToElement(MobileElement element) {

        for (int scroll = 0; scroll < MAX_SCROLL; scroll++) {
            try {
                if (doesElementExist(element, MIN_WAIT)) {
                    scroll = MAX_SCROLL;
                }
            } catch (Exception e) {
                fastScrollDownTouchAction();
            }
        }
    }

    /**
     * Scroll slow to bottom.
     *
     * @param element
     * @return the element
     */
    public MobileElement scrollSlowToElement(MobileElement
    element) {

        for (int scroll = 0; scroll < MAX_SCROLL; scroll++) {
            try {
                if (element.isDisplayed()) {
                    return element;
                } else {
                    slowScrollDownTouchAction();
                }
            } catch (NoSuchElementException e) {
                slowScrollDownTouchAction();
            }

        }
```

```
    return null;
}

/**
 * Scroll to a given text.
 *
 * @param text
 * @return An AndroidElement instance.
 */
public MobileElement scrollToText(String text) {
    if (getSession().getCustomProperties().
    get("isAndroid").equals("true")) return
    scrollToTextAndroid(text);
    else if (getSession().getCustomProperties().get("isIos").
    equals("true")) return scrollToTextIos(text);
    return null;
}

/**
 * Scroll to a given text.
 *
 * @param text
 * @return An AndroidElement instance.
 */
public MobileElement scrollToTextAndroid(String text) {
    String automator = "new UiScrollable(new UiSelector().
    scrollable(true))"
            + ".scrollIntoView(new UiSelector().
            text(\"%s\"))";
    return (AndroidElement) getWait().until(
            visibilityOfElementLocated(AndroidUIAutomator
            (format(automator, text))));
}
```

```
    /**
     * Scroll to text.
     *
     * @param text
     * @return the element
     */
    public MobileElement scrollToTextIos(String text) {
        MobileElement element =
                getSession().getAppiumDriver().
                findElementByXPath(text);
        IntStream.range(0, MAX_SCROLL).forEach(i -> {
            if (!element.isDisplayed()) {
                scroll(Direction.DOWN);
            }
        });
        return element;
    }

    /**
     * Restart iOS/Android App.
     */

    public void restartApp() {
        if (getSession().getCustomProperties().get("isIos").
        equals("true"))
            restartAppIos();
        else if (getSession().getCustomProperties().
        get("isAndroid").equals("true"))
            restartAppAndroid();
        else
            log.error("Error: Wrong Platform!");
    }
```

```
/**
 * Restart the iOS app.
 */
private void restartAppIos() {
    getSession().getAppiumDriver().closeApp();
    getSession().getAppiumDriver().launchApp();
}

/**
 * Restart the Android app.
 */
@Step("Restart the Android application.")
private void restartAppAndroid() {
    if (getSession().getCustomProperties().
    get("defaultService").equals("yes")) {
        getSession().getAppiumDriver().launchApp();
        getSession().getAppiumDriver().
        activateApp(getSession().getCustomProperties().
        get("appPackage"));
    } else {
        try {
            getSession().getAppiumDriver().
            activateApp(getSession().getCustomProperties().
            get("appPackage"));
        } catch (Exception e) {
            AndroidDriver<MobileElement> androidDriver =
            (AndroidDriver<MobileElement>) getSession().
            getAppiumDriver();
            getSession().getAppiumDriver().closeApp();
            Activity activity = new Activity(getSession().
            getCustomProperties().get("appPackage"),
```

```
                    getSession().getCustomProperties().
                    get("appActivity"));
            activity.setAppWaitActivity("com.xxxx");
            androidDriver.startActivity(activity);
        }
    }
}

/**
 * Restart the Session.
 */
@Step("Restart the Session")
public void restartSession() {
    if (getSession().getAppiumDriver().terminateApp("com.
    android.settings")) {
        getSession().getAppiumDriver().
        activateApp(getSession().getCustomProperties().
        get("appPackage"));
    } else {
        AndroidDriver<MobileElement> androidDriver =
        (AndroidDriver<MobileElement>) getSession().
        getAppiumDriver();
        getSession().getAppiumDriver().closeApp();
        Activity activity = new Activity(getSession().
        getCustomProperties().get("appPackage"),
                getSession().getCustomProperties().
                get("appActivity"));
        activity.setAppWaitActivity("com.xxxx");
        androidDriver.startActivity(activity);
    }
}
```

```
/**
 * Takes a screenshot of the current screen
 * and writes it into the project path.
 */
public void takeScreenShot() throws IOException {
    File scrFile = ((TakesScreenshot) getSession().
    getAppiumDriver()).getScreenshotAs(OutputType.FILE);
    BufferedImage image = ImageIO.read(scrFile);
    File outputFile = new File("screenshot.png");
    ImageIO.write(image, "png", outputFile);
}

/**
 * Takes a screenshot of the current screen
 * <p>
 * and writes it into the project path with a given
 file Name.
 */
public void takeScreenShot(String fileName) throws
IOException {
    File scrFile = ((TakesScreenshot) getSession().
    getAppiumDriver()).getScreenshotAs(OutputType.FILE);
    BufferedImage image = ImageIO.read(scrFile);
    File outputFile = new File(fileName);
    ImageIO.write(image, "png", outputFile);
}

/**
 * Takes a screenshot of the current screen
 * <p>
 * and writes it into the project path with a given file
 name after converting to pdf.
```

```
     */
    public void takeScreenShotPdf(String fileName) throws
    IOException {
        File scrFile = ((TakesScreenshot) getSession().
        getAppiumDriver()).getScreenshotAs(OutputType.FILE);
        BufferedImage image = ImageIO.read(scrFile);
        File outputFile = new File(fileName);
        ImageIO.write(image, "png", outputFile);
        String name = fileName.substring(fileName.
        lastIndexOf("/") + 1, fileName.indexOf("."));
        PdfUtil.getPdfFromImage(fileName, "test-output/
        PdfReport/" + name + ".pdf");
    }

    /**
     * Takes a page back.
     */
    public void navigateBack() {
        getSession().getAppiumDriver().navigate().back();
    }

    /**
     * Validate the Background Color in graph.
     *
     * @param element The element
     * @param number  The increment distance from Y center
     * @param color   The expected color
     * @return True if Background Color is displayed ,
     otherwise returns
     * false.
     */
```

```
@Step("Verifying if Background Color is Displayed
in graph")
public boolean isBackGroundColorDisplayed(MobileElement
element, double number, String color) throws IOException {
    int x1 = getXCenter(element);
    int y = getYCenter(element);
    int y1 = (int) (y + (getHeight(element) * number));
    File scrFile = ((TakesScreenshot) getSession().
    getAppiumDriver()).getScreenshotAs(OutputType.FILE);
    BufferedImage image = ImageIO.read(scrFile);
    int clr = image.getRGB(x1, y1);
    int red = (clr & 0x00ff0000) >> 16;
    int green = (clr & 0x0000ff00) >> 8;
    int blue = clr & 0x000000ff;
    if (color.equals("blue"))
        return ((blue > 200) && (red < 100) && (green
        < 100));
    if (color.equals("green"))
        return ((green > 200) && (red < 100) && (blue
        < 100));
    if (color.equals("red"))
        return ((red > 200) && (green < 100) && (blue
        < 100));
    else return false;
}

/**
 * get Coordinates to scroll in wheel.
 *
 * @param direction
 * @param wheel
 * @return @instance Point
```

```
     */
    public Point getScrollCoordinates(Direction direction,
    MobileElement wheel) {
        int adjustedWheelHeight = wheel.getSize().getHeight();
        int xPoint = wheel.getCenter().getX() -
        adjustedWheelHeight;
        int yPoint = wheel.getCenter().getY() +
        adjustedWheelHeight;
        if (direction.equals(Direction.DOWN)) {
            xPoint = wheel.getCenter().getX() +
            adjustedWheelHeight;
        }
        return new Point(xPoint, yPoint);
    }

    public void scrollUpFromCoordinates(MobileElement wheel) {
        Point point = getScrollCoordinates(Direction.
        UP, wheel);
        scrollFromPoint(point, Direction.UP);
    }

    public void scrollDownFromCoordinates(MobileElement
    wheel) {
        Point point = getScrollCoordinates(Direction.
        DOWN, wheel);
        scrollFromPoint(point, Direction.DOWN);
    }

    public void scrollFastUpFromCoordinates(MobileElement wheel) {
        Point point = getScrollCoordinates(Direction.UP, wheel);
        scrollFastFromPoint(point, Direction.UP);
    }
```

```
public void scrollFastDownFromCoordinates(MobileElement
wheel) {
    Point point = getScrollCoordinates(Direction.DOWN, wheel);
    scrollFastFromPoint(point, Direction.DOWN);
}

/**
 * Scroll in wheel from a Point.
 *
 * @param direction
 * @param point
 */
public void scrollFromPoint(Point point, Direction
direction) {
    Dimension size = getSession().getAppiumDriver().
    manage().window().getSize();
    if (direction.equals(Direction.UP)) {
        int width = point.getX();
        int start = point.getY();
        int end = (int) (point.getY() * 0.9);
        touchAction(width, start, end);
    } else {
        int width = point.getX();
        int start = point.getY();
        int end = (int) (point.getY() * 1.1);
        touchAction(width, start, end);
    }
}

/**
 * Scroll in wheel from a Point.
```

```
     *
     * @param direction
     * @param point
     */
    public void scrollFastFromPoint(Point point, Direction
    direction) {
        Dimension size = getSession().getAppiumDriver().
        manage().window().getSize();
        if (direction.equals(Direction.UP)) {
            int width = point.getX();
            int start = point.getY();
            int end = (int) (point.getY() * 0.7);
            touchAction(width, start, end);
        } else {
            int width = point.getX();
            int start = point.getY();
            int end = (int) (point.getY() * 1.3);
            touchAction(width, start, end);
        }
    }

    /**
     * Scroll on a wheel.
     *
     * @param scrollText
     * @param targetText
     */
    public void scrollOnWheelFromText(String scrollText, String
    targetText, Direction direction) {
        int cycle = 0;
        boolean elementFound = false;
        while (!elementFound && cycle < 10) {
```

```
        if (direction.equals(Direction.DOWN)) {
            scrollDownFromCoordinates(getSession().
            getAppiumDriver().findElementByXPath("//*
            [contains(@value,'" + scrollText + "')]"));
        } else {
            scrollUpFromCoordinates(getSession().
            getAppiumDriver().findElementByXPath
            ("//*[contains(@value,'" + scrollText + "')]"));
        }
        try {
            elementFound = doesElementExist(getSession().
            getAppiumDriver().findElementByXPath("//*
            [contains(@value,'" + targetText + "')]"),
            MIN_WAIT);
        } catch (Exception e) {
            e.printStackTrace();
        }
        if (elementFound)
            tapOn(getSession().getAppiumDriver().
            findElementByXPath("//*[contains(@value,'" +
            targetText + "')]"));
        cycle++;
    }
}

/**
 * Scroll on a wheel.
 *
 * @param scrollText
 * @param targetText
 */
```

```
    public void scrollFastOnWheelFromText(String scrollText,
    String targetText, Direction direction) {
        int cycle = 0;
        boolean elementFound = false;
        while (!elementFound && cycle < 10) {
            if (direction.equals(Direction.DOWN)) {
                scrollFastDownFromCoordinates(getSession().
                getAppiumDriver().findElementByXPath("//*
                [contains(@value,'" + scrollText + "')]"));
            } else {
                scrollFastUpFromCoordinates(getSession().
                getAppiumDriver().findElementByXPath("//*
                [contains(@value,'" + scrollText + "')]"));
            }
            try {
                elementFound = doesElementExist(getSession().
                getAppiumDriver().findElementByXPath("//*
                [contains(@value,'" + targetText + "')]"),
                MIN_WAIT);
            } catch (Exception e) {
                e.printStackTrace();
            }
            if (elementFound)
                tapOn(getSession().getAppiumDriver().find
                ElementByXPath("//*[contains(@value,'" +
                targetText + "')]"));
            cycle++;
        }
    }
}
```

`MobileBaseActionScreen`, being what it is, will be extended by all page objects. That way, we will pass the `Session` class reference to all page objects. This will in turn enable the page objects to access the Appium driver for screen-level interactions. The other page objects will need to call the `getSession()` method of the `MobileBaseActionScreen` class to get access to the Appium driver.

The first five wait variables (`MIN_WAIT`, `SMALL_WAIT`, `MED_WAIT`, `LONG_WAIT`, `MAX_WAIT`) are used to specify wait times. We will use these variables in various methods that use timeout parameters, like `doesElementExist`, or various wait methods.

`MAX_SCROLL` is used to specify a limit on scrolling to avoid getting into an infinite loop. The next four variables are for scenarios where the user wants to create an account in one test method and use the same account in other methods. I am assuming these four variables are required for account creation. So, whenever you want to create an account from a specific page object, you can just assign these values once and all other page objects can use these values. We will discuss this in more detail when we write page objects in Chapter 7. But for now, let's discuss the most common screen actions.

Coding for Common Screen Actions

The method `getSoftAssertions` in the `MobileBaseActionScreen` class (Listing 6-4) returns an instance of the `SoftAssertions` class. We will use the `doesElementExist` method in our page objects to check for the existence of various mobile elements. The `waitOn` and `getWait` methods return a `FluentWait` instance. The next five methods are for touch and tap actions on screen.

The next three methods are common wait actions. When using the wait parameters in the methods, we have to use `MIN_WAIT` wherever possible to reduce test execution time. The next ten methods help

to find element location and dimension and can be useful. The methods `checkIfFirstElementIsDisplayedBelowSecondElement` and `isElementRightSideOfOther` can be used to verify the relative position of two elements with respect to one another.

The `sendAppToBackground` method uses the Appium `java_client` method to push the app to the background for a defined amount of time. I encourage you to look at other methods of the `InteractsWithApps` interface. We have used `closeApp`, `launchApp`, `activateApp`, and `terminateApp` in restart methods. We will also use some of these in our test suite (Chapter 8).

The next three methods are quite self-explanatory and deal with click and type actions. These are followed by several swipe and scroll action methods. You can see that I have used some double numbers in the methods. These are for illustration only. I suggest you define constants like wait times instead and use these variables instead of using numbers in these methods. Please note the `scrollToTextAndroid` method, which uses the `UiSelector` and `UiScrollable` classes of the Android `UiAutomator`.

I have added three methods for taking a screenshot. The first one stores the screenshot in the project root in a defined file name. The second method is generic and stores the screenshot in a parameterized location. The third method saves the screenshot in a PDF file. The method `navigateBack` is self-explanatory.

I have added the method `isBackGroundColorDisplayed` to check the background color of a mobile element. I have added a condition for only red, blue, and green and assumed, for any color, that its integer value is greater than 200. I suggest you update this method as per the requirements.

The following methods are swipes on a wheel element. I have added methods for swiping both from a point and from a text. Swiping from text assumes the text location does not change. The text could be part of a bigger string, and that's why I have used the `contains` keyword in the locator.

Summary

In this chapter, we learned how to code common page object actions and variables. We also learned that our page objects can extend the `MobileBaseActionScreen` class to get access to the driver object. In the next chapter, we will see how page objects initialize their elements with this driver object, perform page navigation and assertions, and utilize the reusable methods in the super class.

CHAPTER 7

Creating Page Objects

In the previous chapter, we created the `MobileBaseActionScreen` class, which captures common page object actions and variables. In this chapter, we will create the page objects required to test a typical application. In the next section, we will lay out the page object classes and a workflow class for page navigation.

Initializing Page Objects and Workflow Class

Let's assume our mobile application has three screens. The first screen when we launch the app is the **Account Creation screen**, as shown in Listing 7-1, which, as the name suggests, is the screen where the user enters details to create an account. Let's assume that the required details are **First Name**, **Last Name**, **Email ID**, and **Password** and the user is required to confirm their password. The password must be at least ten characters long with at least two uppercase, two lowercase, and two numeric values. Once the user enters these values and clicks the **Create Account** button, the account is created, and the user is taken to the **App Login screen**, as shown in Listing 7-2. In this screen, the user will enter their email and password and click the **Sign In** button. The user will then be taken to the **About App screen**, as shown in Listing 7-3, which shows some documentation on the app. We will also need a workflow class,

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_7

as shown in Listing 7-4, for screen navigation in our test suite. This will especially be required when there are dozens of page objects and we need a landing screen for every test suite. Let's call this class `ScreenNavigation`.

Now let's create these three page objects within the `screens` folder and the `workflow` class in the `workflows` folder so that our folder structure looks like that in Figure 7-1.

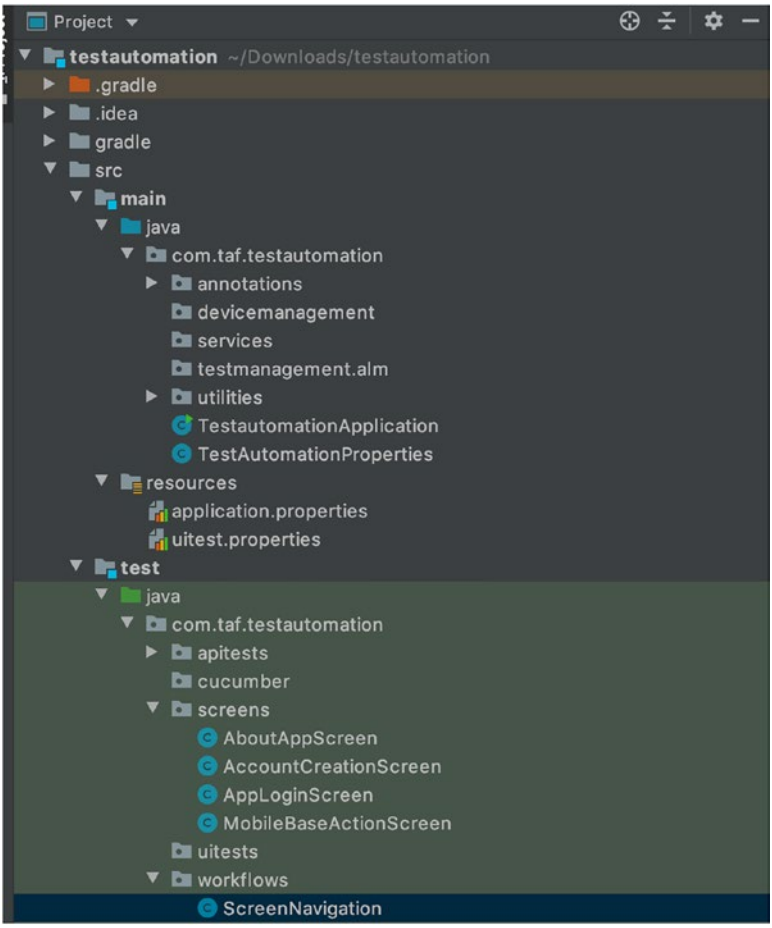


Figure 7-1. *Folder structure with screens and workflows folders*

The code snippets for the screens and workflow class are as follows:

Listing 7-1. AccountCreationScreen.java

```
package com.taf.testautomation.screens;

import com.taf.testautomation.Session;
import io.appium.java_client.MobileElement;
import io.appium.java_client.pagefactory.*;
import io.qameta.allure.Step;
import org.apache.commons.lang3.RandomStringUtils;
import org.openqa.selenium.support.PageFactory;

public class AccountCreationScreen extends
MobileBaseActionScreen {

    public AccountCreationScreen(Session session) {
        super(session);
        PageFactory.initElements(new
        AppiumFieldDecorator(session.getAppiumDriver()), this);
    }

    private String XPATH_ANDROID = "", XPATH_IOS = "";

    @AndroidFindBy(xpath = "//android.widget.EditText
    [@text='xxxx']")
    @iOSXCUITFindBy(accessibility = "xxxx")
    private MobileElement firstNameField;

    @AndroidFindBy(xpath = "//android.widget.EditText
    [@text='xxxx']")
    @iOSXCUITFindBy(accessibility = "xxxx")
    private MobileElement lastNameField;

    @AndroidFindBy(xpath = "//android.widget.EditText
    [@text='xxxx']")
```

```
    @iOSXCUITFindBy(accessibility = "xxxx")
    private MobileElement emailIdField;

   @AndroidFindBy(xpath = "//android.widget.EditText
   [@text='xxxx']")
    @iOSXCUITFindBy(accessibility = "xxxx")
    private MobileElement passwordField;

    @AndroidFindBy(xpath = "//android.widget.EditText
    [@text='xxxx']")
    @iOSXCUITFindBy(accessibility = "xxxx")
    private MobileElement confirmPasswordField;

    @AndroidFindBy(xpath = "//android.widget.Button
    [@text='xxxx']")
    @iOSXCUITFindBy(accessibility = "xxxx")
    private MobileElement createAccountButton;

    @HowToUseLocators(androidAutomation =
    LocatorGroupStrategy.ALL_POSSIBLE,
            iOSXCUITAutomation = LocatorGroupStrategy.ALL_
            POSSIBLE)
    @AndroidFindBy(accessibility = "xxxx")
    @AndroidFindBy(xpath = "//android.widget.Button
    [@text='xxxx']")
    @iOSXCUITFindBy(accessibility = "xxxx")
    @iOSXCUITFindBy(xpath = "//XCUIElementTypeButton
    [@name='xxxx']")
    private MobileElement okButton;

    /**
     * Get AppLoginScreen instance
     *
     * @return {@link AppLoginScreen}
     */
```

```
    @Step("Create account with firstNam, lastName, email and
    password")
    public AppLoginScreen createAccount() {
        firstName = RandomStringUtils.randomAlphanumeric(10);
        lastName = RandomStringUtils.randomAlphanumeric(10);
        email = RandomStringUtils.randomAlphanumeric(10) +
        "@gmail.com";
        password = RandomStringUtils.randomAlphanumeric(4) +
        RandomStringUtils.randomNumeric(2)
        + RandomStringUtils.random(2).toLowerCase() +
        RandomStringUtils.random(2).toUpperCase();
        if (getSession().getCustomProperties().
        get("isAndroid").equals("true")) {
            type(firstNameField, firstName);
            type(lastNameField, lastName);
            type(emailIdField, email);
            type(passwordField, password);
        } else {
            clickAndType(firstNameField, firstName);
            clickAndType(lastNameField, lastName);
            clickAndType(emailIdField, email);
            clickAndType(passwordField, password);
            getSession().getAppiumDriver().hideKeyboard();
        }
        click(createAccountButton);
        if (doesElementExist(okButton, MIN_WAIT)) {
            click(okButton);
        }
        waitInSeconds(SMALL_WAIT);
        return new AppLoginScreen(getSession());
    }
}
```

Listing 7-2. AppLoginScreen.java

```
package com.taf.testautomation.screens;

import com.taf.testautomation.Session;
import io.appium.java_client.MobileElement;
import io.appium.java_client.pagefactory.*;
import io.qameta.allure.Step;
import org.openqa.selenium.support.PageFactory;

public class AppLoginScreen extends MobileBaseActionScreen {

    public AppLoginScreen(Session session) {
        super(session);
        PageFactory.initElements(new
        AppiumFieldDecorator(session.getAppiumDriver()), this);
    }

    private String XPATH_ANDROID = "", XPATH_IOS = "";

    @AndroidFindBy(id = "xxxx")
    @iOSXCUITFindBy(accessibility = "xxxx")
    private MobileElement emailField;

    @AndroidFindBy(xpath = "//android.widget.EditText
    [@text='xxxx']")
    @iOSXCUITFindBy(accessibility = "xxxx")
    private MobileElement passwordField;

    @HowToUseLocators(androidAutomation =
    LocatorGroupStrategy.ALL_POSSIBLE,
            iOSXCUITAutomation = LocatorGroupStrategy.ALL_
            POSSIBLE)
    @AndroidFindBy(id = "xxxx")
```

```
@AndroidFindBy(xpath = "//android.widget.Button[contains
(@resource-id,'xxxx')]")
@iOSXCUITFindBy(accessibility = "xxxx")
@iOSXCUITFindBy(xpath = "//XCUIElementTypeButton[@
name='xxxx']")
private MobileElement signInButton;

/**
 * Get AboutAppScreen after login to App with new account
 *
 * @return {@link AboutAppScreen}
 */
@Step("Sign-in to App with new account")
public AboutAppScreen signInToApp() {
    if (getSession().getCustomProperties().
    get("isAndroid").equals("true")) {
        type(emailField, email);
        type(passwordField, password);
    } else {
        clickAndType(emailField, email);
        clickAndType(passwordField, password);
        getSession().getAppiumDriver().hideKeyboard();
    }
    click(signInButton);
    waitInSeconds(SMALL_WAIT);
    return new AboutAppScreen(getSession());
}

/**
 * Get AboutAppScreen after login to App with existing
 account from properties file
 *
```

```
     * @return {@link AboutAppScreen}
     */
    @Step("Sign-in to App with existing account")
    public AboutAppScreen signInToApp(String email, String
    password) {
        if (getSession().getCustomProperties().
        get("isAndroid").equals("true")) {
            type(emailField, email);
            type(passwordField, password);
        } else {
            clickAndType(emailField, email);
            clickAndType(passwordField, password);
            getSession().getAppiumDriver().hideKeyboard();
        }
        click(signInButton);
        waitInSeconds(SMALL_WAIT);
        return new AboutAppScreen(getSession());
    }
}
```

Listing 7-3. AboutAppScreen.java

```
package com.taf.testautomation.screens;

import com.taf.testautomation.Session;
import io.appium.java_client.MobileElement;
import io.appium.java_client.pagefactory.*;
import io.qameta.allure.Step;
import org.openqa.selenium.NoSuchElementException;
import org.openqa.selenium.support.PageFactory;

import java.util.List;
import java.util.stream.Collectors;
```

```
import java.util.stream.IntStream;
import java.util.stream.Stream;

public class AboutAppScreen extends MobileBaseActionScreen {

    public AboutAppScreen(Session session) {
        super(session);
        PageFactory.initElements(new
        AppiumFieldDecorator(session.getAppiumDriver()), this);
    }

    private String XPATH_ANDROID = "", XPATH_IOS = "";

    @AndroidFindBy(xpath = "//android.widget.TextView
    [@text='xxxx']")
    @iOSXCUITFindBy(xpath = "//XCUIElementTypeStaticText
    [@name='xxxx'][@label='xxxx']")
    private MobileElement screenTitle;

    @AndroidFindBy(accessibility = "xxxx")
    @iOSXCUITFindBy(accessibility = "xxxx")
    private MobileElement appVersion;

    @HowToUseLocators(androidAutomation =
    LocatorGroupStrategy.ALL_POSSIBLE,
            iOSXCUITAutomation = LocatorGroupStrategy.ALL_
            POSSIBLE)
    @AndroidFindBy(xpath = "//android.widget.TextView
    [@text='xxxx']")
    @AndroidFindBy(accessibility = "xxxx")
    @iOSXCUITFindBy(xpath = "//XCUIElementTypeStaticText
    [@name='xxxx']")
    @iOSXCUITFindBy(accessibility = "xxxx")
    private MobileElement appName;
```

```
@AndroidFindBy(xpath = "//android.widget.TextView
[@text='xxxx' or @text='xxxx']")
@iOSXCUITFindBy(xpath = "//XCUIElementTypeStaticText
[contains(@name,'xxxx') or contains(@value,'xxxx')]")
private MobileElement copyrightText;

@AndroidFindBy(xpath = "//android.widget.ImageView
[@index='xxxx']")
@iOSXCUITFindBy(xpath = "//XCUIElementTypeImage[contains
(@name,'xxxx')]")
private MobileElement appLogo;

@AndroidFindBy(xpath = "//android.widget.ImageView")
@iOSXCUITFindBy(xpath = "//XCUIElementTypeImage")
private List<MobileElement> appImages;

/**
 * Validate if Screen Title is displayed
 *
 * @return True if Screen Title is displayed ,
 otherwise returns
 * false.
 */
@Step("Verifying the Screen Title is Displayed")
public boolean isScreenTitleDisplayed() {
    return doesElementExist(screenTitle, SMALL_WAIT);
}

/**
 * Validate if App logo is displayed
 *
 * @return True if App logo is displayed, otherwise returns
 * false.
 */
```

```
@Step("Verifying the App logo is Displayed")
public boolean isAppLogoDisplayed() {
    fastScrollDownTouchAction();
    if (getSession().getCustomProperties().
    get("isAndroid").equals("true")) {
        return doesElementExist(appLogo, SMALL_WAIT);
    } else {
        return appLogo.isEnabled();
    }
}

/**
 * Validate if App Name is displayed
 *
 * @return True if App Name is displayed, otherwise returns
 * false.
 */
@Step("Verifying the App Name is Displayed")
public boolean isAppNameDisplayed() {
    fastScrollDownTouchAction();
    return doesElementExist(appName, SMALL_WAIT) &&
    doesElementExist(appName, MIN_WAIT);
}

/**
 * Validate if App Version is displayed
 *
 * @return True if App Version is displayed,
 otherwise returns
 * false.
 */
@Step("Verifying the App Version is Displayed")
```

```
public boolean isAppVersionDisplayed(String text) {
    IntStream.range(0, 2).forEach(i -> scrollUpTouchAction());
    waitInSeconds(MIN_WAIT);
    return doesElementExist(appVersion, SMALL_WAIT) &&
    appVersion.getText().contains(text);
}

/**
 * Validate if Copyright text is displayed
 *
 * @return True if Copyright text is displayed,
 otherwise returns
 * false.
 */
@Step("Verifying the Copyright text is Displayed")
public boolean isCopyrightTextDisplayed(String str) {
    setLocatorText(str);
    if (getSession().getCustomProperties().
    get("isAndroid").equals("true")) {
        try {
            return doesElementExist(getSession().
            getAppiumDriver().findElementByXPath(XPATH_
            ANDROID), MIN_WAIT);
        } catch (NoSuchElementException e) {
            return false;
        }
    } else {
        try {
            fastScrollDownTouchAction();
            return doesElementExist(getSession().
            getAppiumDriver().
            findElementByXPath(XPATH_IOS), MIN_WAIT);
```

```
        } catch (NoSuchElementException e) {
            return false;
        }
    }
}

/**
 * Validate if App Description is displayed
 *
 * @return True if App Description is displayed,
 otherwise returns
 * false.
 */
@Step("Verifying the App Description is Displayed")
public boolean isAppDescriptionDisplayed(String str1,
String str2) {
    boolean result = true;
    Stream<String> locatorTextStream =
    Stream.of(str1, str2);
    List<String> locatorTextList = locatorTextStream.collect
    (Collectors.toList());
    for (String str : locatorTextList) {
        setLocatorText(str);
        if (getSession().getCustomProperties().
        get("isAndroid").equals("true")) {
            result = result && doesElementExist
            (getSession().getAppiumDriver().
            findElementByXPath(XPATH_ANDROID), MIN_WAIT);
        } else {
            result = result && doesElementExist
            (getSession().getAppiumDriver().
            findElementByXPath(XPATH_IOS), MIN_WAIT);
```

```
            }
        }
        return result;
    }

    /**
     * Validate if app images are displayed
     *
     * @return True if app images are displayed,
     otherwise returns
     * false.
     */
    @Step("Verifying the app images are Displayed")
    public boolean areAppImagesDisplayed() {
        return doesElementExist(appImages.get(0), MIN_WAIT)
        && doesElementExist(appImages.get(1), MIN_WAIT) &&
        doesElementExist(appImages.get(2), MIN_WAIT);
    }

    private void setLocatorText(String text) {
        XPATH_ANDROID = "//android.widget.TextView[@text='" +
        text + "']";
        XPATH_IOS = "//XCUIElementTypeStaticText[@name='" +
        text + "']";
    }
}
```

Listing 7-4. ScreenNavigation.java

```
package com.taf.testautomation.workflows;

import com.taf.testautomation.Session;
import com.taf.testautomation.screens.AboutAppScreen;
import com.taf.testautomation.screens.AccountCreationScreen;
```

```
import com.taf.testautomation.screens.MobileBaseActionScreen;

public class ScreenNavigation extends MobileBaseActionScreen {

    public ScreenNavigation(Session session) {
        super(session);
    }

    /**
     * Navigate to AboutAppScreen after creating account
     *
     * @param username
     * @param password
     * @return {@link AboutAppScreen}
     */
    public AboutAppScreen getAboutAppScreenFromAccountCreation
    Screen(String username, String password) {
        AccountCreationScreen accountCreationScreen = new
        AccountCreationScreen(getSession());
        if (getSession().getCustomProperties().
        get("isAndroid").equals("true")) {
            return accountCreationScreen.createAccount().
            signInToApp(username, password);
        } else {
            return accountCreationScreen
                    .createAccount()
                    .signInToApp();
        }
    }
}
```

You can see that the page objects and the `workflow` class extend the `MobileBaseActionScreen` class and have access to the `Session` object through its constructor. This is what we discussed in the previous chapter when creating the `MobileBaseActionScreen` class. The `initElements` method of the `PageFactory` class is used to initialize the locators in the page object. In the next section, I will explain our locator strategy.

Deciding on Locator Strategy

In this framework, I have used `appium.java_client` annotations, namely `@AndroidFindBy` and `@iOSXCUITFindBy`, as the locator strategy. The advantage of this is that we have a single variable for both Android and iOS, which helps in writing a common test suite for both platforms.

I have used various locators, like accessibility, ID, and xpath, to give you a flavor of the various options available to you. Also, while forming xpath, I have used various class names and attributes. I have used multiple attributes in a single xpath by using the `or` keyword. I have also created a list of mobile elements. Finally, I have used the `@HowToUseLocators` annotation, as shown in Listing 7-5, from the `appium.java_client` to define multiple locators for the same mobile element. In the next section, we will further discuss the semantics of the page object methods.

Listing 7-5. @HowToUseLocators Annotation

```
@HowToUseLocators(androidAutomation = LocatorGroupStrategy.ALL_
POSSIBLE,
        iOSXCUITAutomation = LocatorGroupStrategy.ALL_POSSIBLE)
```

Writing Page Object Methods

You will notice that there are two types of methods in the page object. The ones that return Booleans are fed into the soft assertion in the test suite. We will discuss this in more detail in the next chapter. The other methods that return page objects are used in screen navigation. Of course, you will need several other methods with either void or other return types in real-life projects. In addition, you will use private methods in real-life projects that perform some screen actions to be used in other, public methods. I am using the `@Step` Allure annotation to describe the methods. This will be very useful for Allure reporting, as the same step shows up in Allure reports (Chapter 11).

The one method we repeatedly use is the `doesElementExist` method from the `MobileBaseActionScreen` class. We also use the scroll up and scroll down methods from `MobileBaseActionScreen` whenever required. We also dynamically create locators using the `setLocatorText` method in the `isCopyrightTextDisplayed` and `isAppDescriptionDisplayed` methods in the `AboutAppScreen` class. We will use these methods for localization testing (Chapter 19).

Let's also observe the `createAccount` method in `AccountCreationScreen` and recall that we defined the static variables `firstName`, `lastName`, `email`, and `password` in the `MobileBaseActionScreen` class. When we assign values to these variables from any page object, the same values are propagated to all page objects. As a result, the first `signInToApp` method with no parameters in `AppLoginScreen` can still access email and password values created in previous step.

Please notice that any page object can get access to the driver by calling the `getSession().getAppiumDriver()` method. Subsequently, we can call any method using this driver instance. For example, in the `isCopyrightTextDisplayed` method in the `AboutAppScreen` class, we use

the `getSession().getAppiumDriver().findElementByXPath("xxxx")` method to locate the element. Similarly, we use the `getSession().getAppiumDriver().hideKeyboard()` method in the App Login screen to hide the keyboard.

Finally, notice the `getAboutAppScreenFromAccountCreationScreen(String username, String password)` method to see how we use the `workflow` class to navigate to a landing screen. We will use this method in our test suite in the next chapter.

Summary

In this chapter, we learned how to create page objects and initialize the page elements. In addition, we discussed locator strategy and page object methods. We also created a `workflow` class for page navigation. In the next chapter, we will write and run our first test suite.

CHAPTER 8

Writing Your First Test Suite

In the previous chapter, we created the initial set of page objects and the `workflow` class for screen navigation. In this chapter, we will create a test suite and implement soft assertions. We will also examine the use of various Junit and Allure annotations in test methods and see how to assign these methods to particular Gradle tasks. Let's begin!

Using Various Annotations

Let's create a test suite class named `AboutAppTestSuite`, as shown in Listing 8-2 in the `com/taf/testautomation/uitests` folder, to test the `AboutAppScreen` elements. We will also create a `BaseTest`, as shown in Listing 8-1 within the `src/test/java/com/taf/testautomation` folder, that each test suite will extend. In this `BaseTest`, we will call the `setUp()` and `tearDown()` methods, which will be used by all test suites. In addition, `BaseTest` has a static `dataTable` variable, which we will explain in Chapter 10. So, our folder structure should now look like that in Figure 8-1.

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_8

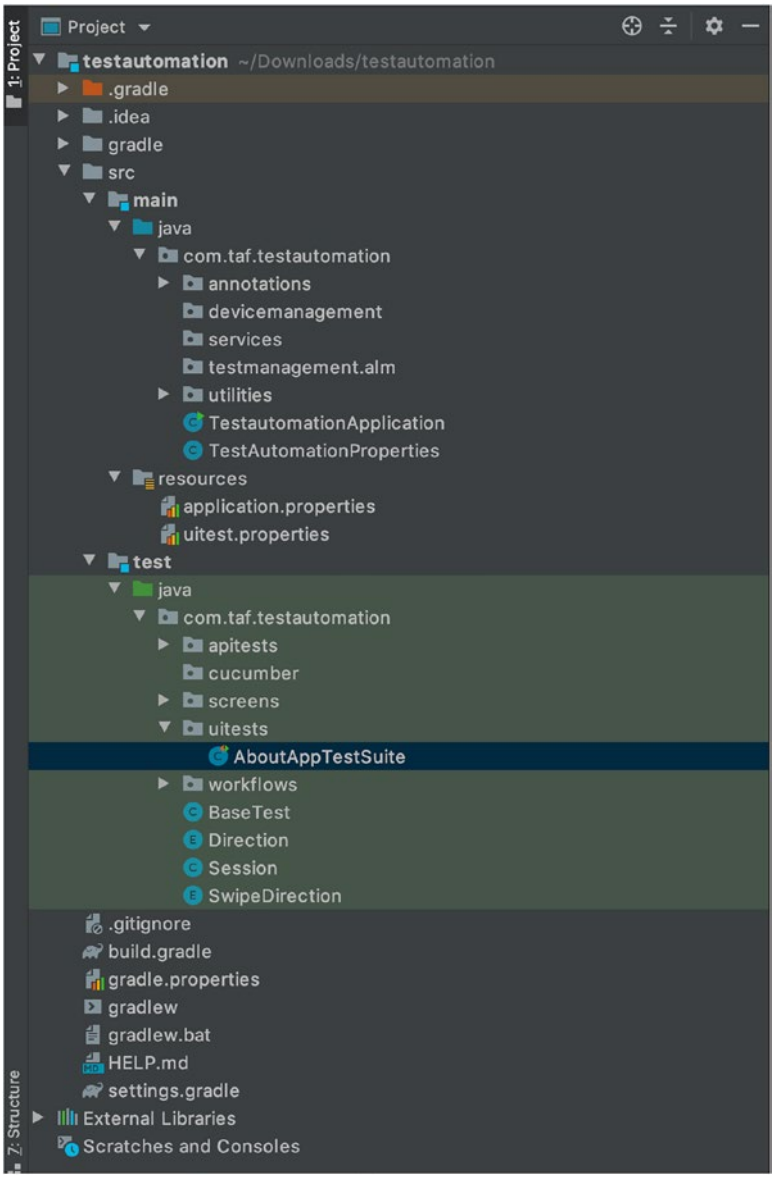


Figure 8-1. *Folder structure with AboutAppTestSuite and BaseTest*

The code snippets for the test classes are as follows. We will enhance this test suite with a reporting module, HP ALM integration, and screenshot capabilities in subsequent chapters.

Listing 8-1. BaseTest.java

```
package com.taf.testautomation;

import lombok.Getter;
import lombok.Setter;
import org.assertj.core.api.SoftAssertions;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.HashMap;

@Getter
@Setter
@SuppressWarnings("rawtypes")
public class BaseTest {

    protected Session session = new Session();
    protected HashMap<String, String> customProperties =
    session.getCustomProperties();
    private Logger logger = LoggerFactory.getLogger(this.
    getClass());
    protected static String[][] dataTable;

    @BeforeAll
    public void setUp() throws Exception {
        log("Initializing Session");
        session = startDefaultSession();
```

```
        log("Session created");
    }

    @AfterAll
    public void tearDown() throws Exception {
        log("Destroying Session");
        closeSession();
        log("Session destroyed");
    }

    public void log(String message) {
        getLogger().info(message);
    }

    public void logError(String message) {
        getLogger().error(message);
    }

    public Session startDefaultSession() throws Exception {
        try {
            session.startSession();
        } catch (Exception e) {
            logError("Error starting Session" +
            e.getMessage());
        }
        if (session.getAppiumDriver() != null) {
            return session;
        } else return startDefaultSession();
    }

    public void closeSession() {
        try {
            session.closeSession();
        } catch (Exception e) {
```

```
            logError("Error closing Session" + e.getMessage());
        }
    }
}
```

Listing 8-2. AboutAppTestSuite.java

```
package com.taf.testautomation.uitests;

import com.taf.testautomation.BaseTest;
import com.taf.testautomation.annotations.*;
import com.taf.testautomation.screens.AboutAppScreen;
import com.taf.testautomation.workflows.ScreenNavigation;
import io.qameta.allure.*;
import org.assertj.core.api.SoftAssertions;
import org.junit.jupiter.api.*;

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
@Epic("xxxx")
@Feature("About App Page Layout")
public class AboutAppTestSuite extends BaseTest {

    private AboutAppScreen aboutAppScreen;
    protected String testStatus = "";
    private static final String SCREEN_NAME = "aboutAppScreen";
    private static int i = 0, j = 0;

    @BeforeAll
    @Override
    public void setUp() throws Exception {
        super.setUp();
    }
```

```
    @BeforeEach
    public void navigateToScreen() {
        log("Navigating to " + SCREEN_NAME);
        if (testStatus.isEmpty()) {
            aboutAppScreen = new ScreenNavigation(session).
            getAboutAppScreenFromAccountCreationScreen
            (getCustomProperties().get("username"),
            getCustomProperties().get("password"));
        } else {
            log("User in About App screen already");
        }

    }

    @AfterAll
    @Override
    public void tearDown() throws Exception {
        super.tearDown();
    }

    @Severity(SeverityLevel.CRITICAL)
    @Issue("xxxx")
    @DisplayName("xxxx")
    @Description("xxxx: Verify that the AboutAppScreen Title is
    displayed")
    @Test
    @Order(1)
    @Smoke
    @Regression
    @SIT
    @AT
    public void testScenario1() {
        String tcName = new Object() {
```

```
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    SoftAssertions.assertSoftly(
            softAssertions -> {
                softAssertions.assertThat(aboutAppScreen.
                isScreenTitleDisplayed()).as("The Screen
                title is displayed").isTrue();
            }
    );
    testStatus = aboutAppScreen.isScreenTitleDisplayed() ?
    "Passed" : "Failed";
    updateTCPassCount();
}

@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the App Logo is displayed")
@Test
@Order(2)
@Smoke
public void testScenario2() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    SoftAssertions.assertSoftly(
            softAssertions -> {
                softAssertions.assertThat(aboutAppScreen.
                isAppLogoDisplayed()).as("The App Logo is
                displayed").isTrue();
            }
    );
```

```
        testStatus = aboutAppScreen.isAppLogoDisplayed() ?
        "Passed" : "Failed";
        updateTCPassCount();
    }

    @Severity(SeverityLevel.CRITICAL)
    @Issue("xxxx")
    @DisplayName("xxxx")
    @Description("xxxx: Verify that the Software Name is
    displayed")
    @Test
    @Order(3)
    @Regression
    public void testScenario3() {
        String tcName = new Object() {
        }.getClass().getEnclosingMethod().getName();
        log("Test Name" + tcName);

        SoftAssertions.assertSoftly(
                softAssertions -> {
                    softAssertions.assertThat(aboutAppScreen.
                    isAppNameDisplayed()).as("The App Name is
                    displayed").isTrue();
                }
        );
        testStatus = aboutAppScreen.isAppNameDisplayed() ?
        "Passed" : "Failed";
        updateTCPassCount();
    }

    @Severity(SeverityLevel.CRITICAL)
    @Issue("xxxx")
    @DisplayName("xxxx")
```

```
@Description("xxxx: Verify that the App Version is displayed")
@Test
@Order(4)
@SIT
public void testScenario4() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    SoftAssertions.assertSoftly(
            softAssertions -> {
                softAssertions.assertThat(aboutAppScreen.
                isAppVersionDisplayed("xxxx")).as("The App
                Version is displayed").isTrue();
            }
    );
    testStatus = aboutAppScreen.
    isAppVersionDisplayed("xxxx") ? "Passed" : "Failed";
    updateTCPassCount();
}

@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the App images are
displayed")
@Test
@Order(5)
@AT
public void testScenario5() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
```

```
        log("Test Name" + tcName);

        SoftAssertions.assertSoftly(
                softAssertions -> {
                    softAssertions.assertThat(aboutAppScreen.
                    areAppImagesDisplayed()).as("The App images
                    are displayed").isTrue();
                }
        );
        testStatus = aboutAppScreen.areAppImagesDisplayed() ?
        "Passed" : "Failed";
        updateTCPassCount();
    }

    private void updateTCPassCount() {
        i++;
        if (testStatus.equals("Passed")) j++;
    }
}
```

`@TestMethodOrder` is a Junit annotation that is used with the `@Order` annotation to define the sequence in which the test methods will be executed. The `@TestInstance` annotation has two modes— `LifeCycle.PER_METHOD`, which is the default mode, and `LifeCycle.PER_CLASS`. Junit creates test instances as per these modes. The `@Test` annotation is used to define a test method in Junit. For a detailed write-up, you should refer to https://www.baeldung.com/junit-testinstance-annotation. The Junit annotation `@DisplayName` is used to assign a custom display name to the test method that the build tool uses. I have also used the `@BeforeAll`, `@BeforeEach`, and `@AfterAll` Junit annotations, which are self-explanatory.

Please note the way in which we should use the Gradle tasks for each test, using the `@Smoke`, `@Regression`, `@SIT`, and `@AT` annotations. These are the annotations we created in `build.gradle` and defined within the annotations folder (Chapter 3). When we run the Gradle tests (refer to last section of this chapter), we need to select a tag, and only the tests that are so tagged will be run. For example, if we run `AboutAppTestSuite` (Listing 8-2) with the `@Smoke` annotation, only test scenarios 1 and 2 would be run.

Writing Soft Assertions

I prefer the `assertSoftly` static method of the `AssertJ` soft assertion. This allows multiple `assert` statements within the same method block. However, you may use any other soft assertions, like the one we defined in the `MobileBaseActionScreen` class, and use the `assertThat` method of that class.

Plugging in the Reporting Module

We will write separate methods for the reporting module for each test case. This module will collect screenshots from each test method, gather device data, and create a PDF report. I will save this for Chapter 13 to reduce information overload. For now, we will see how to run our test methods based on Gradle tasks.

Running Test Suite in Gradle

To run the test suite, simply right-click on it and select the appropriate tag, as seen in Figure 8-2.

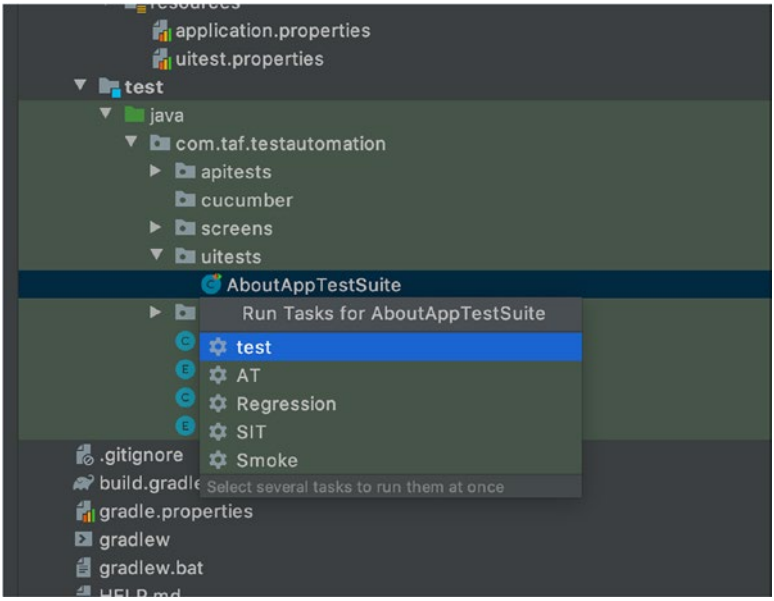


Figure 8-2. *Gradle tasks after right-click on test suite*

The other way to run the test suite is to open Run ➤ Edit Configurations from the Gradle menu bar at the top and then select "task," as shown in Figure 8-3 (`:cleanTest` is optional but good to have).

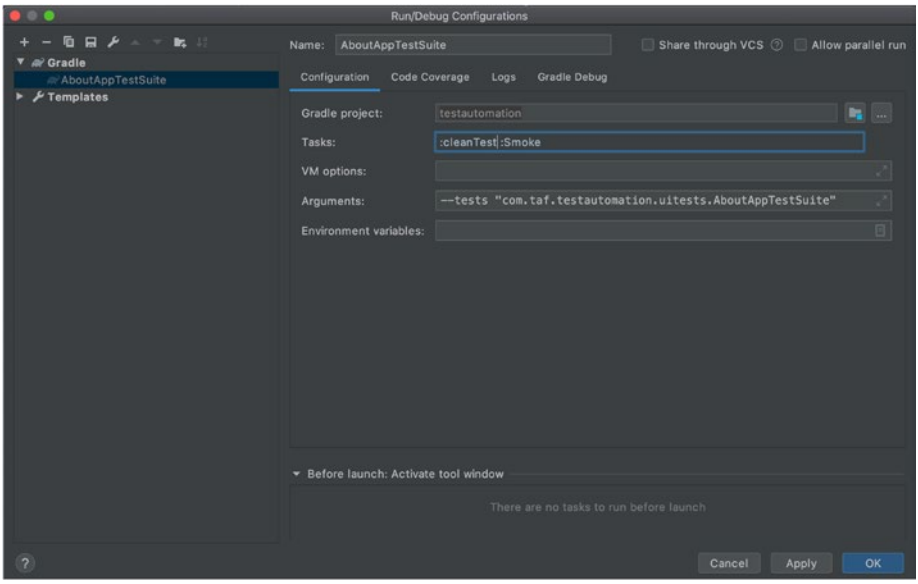


Figure 8-3. *Gradle run configuration*

You can also run the test suite from the Gradle command line, as shown in Figure 8-4.

Figure 8-4. *Using Gradle command line to run test suite*

Summary

In this chapter, we learned how to write test suites using soft assertions and various annotations. We also learned how to run test suites from the IntelliJ run configuration and the command line. In the next chapter, we will discuss how to work with test data in various formats.

CHAPTER 9

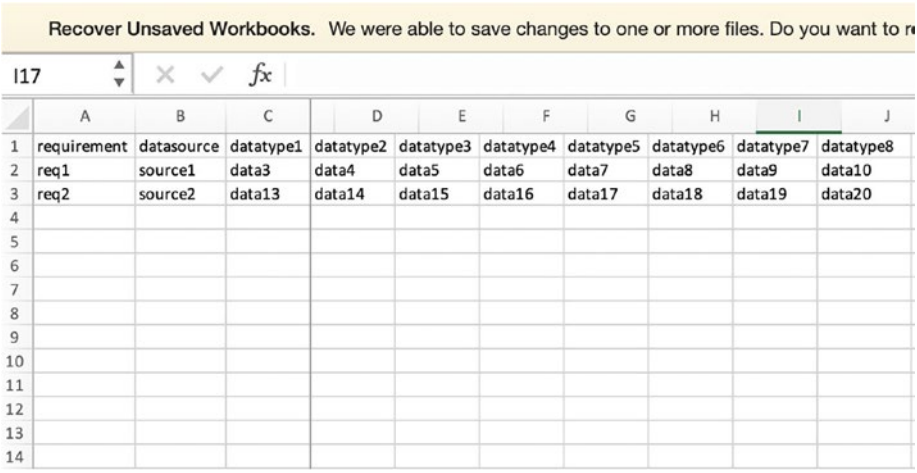
Importing Test Data from Excel, XML, or Other Formats

In the previous chapter, we learned how to write test suites with soft assertions and to execute these with Gradle tasks. In this chapter, we will learn how to work with test data that is stored in formats like Excel, XML, text, and so forth. The test data might be an app configuration file, data table with expected data for given inputs, or simply strings for localization testing (see Chapter 19). Let's start by importing test data from Excel files.

Importing Test Data from Excel

You will recall that in Chapter 4, while creating the `uitests.properties` file, we specified `dataTable1` and `dataTable2` as sources of test data. This is to show you how we can source test data from multiple files. I have created two identical Excel files for demonstration purposes and put those in the `src/test/resources/testdata/excel` folder (screenshot in Figure 19-1). The Excel files look as shown in Figure 9-1. You need to replace this data with actual test data.

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_9


Recover Unsaved Workbooks. We were able to save changes to one or more files. Do you want to r

I17

	A	B	C	D	E	F	G	H	I	J
1	requirement	datasource	datatype1	datatype2	datatype3	datatype4	datatype5	datatype6	datatype7	datatype8
2	req1	source1	data3	data4	data5	data6	data7	data8	data9	data10
3	req2	source2	data13	data14	data15	data16	data17	data18	data19	data20
4										
5										
6										
7										
8										
9										
10										
11										
12										
13										
14										

Figure 9-1. *Excel data source*

Now, let's create a folder named `excelutil` within the `src/main/java/com/taf/testautomation/utilities` folder. Write an `ExcelUtil` class, as shown in Listing 9-1. This class will read data from the test data location and generate separate JSON files with the key being the datatype and value from each subsequent row. So, for the preceding test data there would be two JSON files named `source1TestData1.json` and `source2TestData1.json`. The other test data file (`testData2.xlsx`) would be converted to `req1TestData2.json` and `req2TestData2.json`. Then these JSON files would be saved in the `src/test/resources/testdata/json` folder.

Listing 9-1. ExcelUtil.java

```
package com.taf.testautomation.utilities.excelutil;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonObject;
import org.apache.poi.ss.usermodel.CellType;
import org.apache.poi.ss.usermodel.Row;
```

```
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import java.io.*;
import java.net.URL;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Properties;

public class ExcelUtil {

    private Workbook dataTable;
    private static HashMap<String, String> customProperties =
    new HashMap<>();
    private static final String JSON_LOCATION = "src/test/
    resources/testdata/json/";
    private static final String JSON1_POSTFIX =
    "TestData1.json";
    private static final String JSON2_POSTFIX =
    "TestData2.json";

    /**
     * Populate customProperties map
     * with the value from uitest.properties file
     */
    private static Reader reader;

    public static HashMap<String, String>
    getCustomProperties() {
        ClassLoader loader = ExcelUtil.class.getClassLoader();
        URL myURL = loader.getResource("uitest.properties");
        String path = myURL.getPath();
```

```
        try {
            reader = new FileReader(path);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        Properties prop = new Properties();
        try {
            prop.load(reader);
        } catch (IOException e) {
            e.printStackTrace();
        }
        prop.forEach((k, v) -> customProperties.put(k.
        toString(), v.toString()));
        return customProperties;
    }

    public void generateJsonFilesFromExcel1() throws
    Exception {
        File excelFile = new File(getCustomProperties().
        get("dataTable1"));
        getJsonFromExcel1(excelFile);
    }

    public void generateJsonFilesFromExcel2() throws
    Exception {
        File excelFile = new File(getCustomProperties().
        get("dataTable2"));
        getJsonFromExcel2(excelFile);
    }

    public void getJsonFromExcel1(File excelFile) throws
    Exception {
        dataTable = new XSSFWorkbook(excelFile);
```

```
int sheetNumber = dataTable.getNumberOfSheets() - 1;
ArrayList<String> columnNames = new
ArrayList<String>();
Sheet sheet = dataTable.getSheetAt(sheetNumber);
Iterator<Row> sheetIterator = sheet.iterator();

while (sheetIterator.hasNext()) {
    Row currentRow = sheetIterator.next();
    if (currentRow.getRowNum() != 0) {
        JsonObject jsonObject = new JsonObject();

        for (int j = 0; j < columnNames.size(); j++) {
            if (currentRow.getCell(j) != null) {
                if (currentRow.getCell(j).
                getCellTypeEnum() == CellType.STRING) {
                    jsonObject.addProperty(columnNames.
                    get(j), currentRow.getCell(j).
                    getStringCellValue());
                } else if (currentRow.
                getCell(j).getCellTypeEnum() ==
                CellType.NUMERIC) {
                    jsonObject.addProperty(columnNames.
                    get(j), currentRow.getCell(j).
                    getNumericCellValue());
                } else if (currentRow.
                getCell(j).getCellTypeEnum() ==
                CellType.BOOLEAN) {
                    jsonObject.addProperty(columnNames.
                    get(j), currentRow.getCell(j).
                    getBooleanCellValue());
                } else if (currentRow.getCell(j).
                getCellTypeEnum() == CellType.BLANK) {
```

```
                        jsonObject.addProperty(columnNames.
                        get(j), "");
                    }
                } else {
                    jsonObject.addProperty(columnNames.
                    get(j), "");
                }

            }
            saveJSON(currentRow.getCell(1).
            getStringCellValue().toLowerCase(),
            jsonObject);
        } else if (currentRow.getRowNum() == 0) {
            // store column names
            for (int k = 0; k < currentRow.
            getPhysicalNumberOfCells(); k++) {
                columnNames.add(currentRow.getCell(k).
                getStringCellValue());
            }
        }
    }
}

public void getJsonFromExcel2(File excelFile) throws
Exception {
    dataTable = new XSSFWorkbook(excelFile);
    int sheetNumber = dataTable.getNumberOfSheets() - 1;
    ArrayList<String> columnNames = new
    ArrayList<String>();
    Sheet sheet = dataTable.getSheetAt(sheetNumber);
    Iterator<Row> sheetIterator = sheet.iterator();
```

```
while (sheetIterator.hasNext()) {
    Row currentRow = sheetIterator.next();
    if (currentRow.getRowNum() != 0) {
        JsonObject jsonObject = new JsonObject();

        for (int j = 0; j < columnNames.size(); j++) {
            if (currentRow.getCell(j) != null) {
                if (currentRow.getCell(j).
                getCellTypeEnum() == CellType.STRING) {
                    jsonObject.addProperty(columnNames.
                    get(j), currentRow.getCell(j).
                    getStringCellValue());
                } else if (currentRow.
                getCell(j).getCellTypeEnum() ==
                CellType.NUMERIC) {
                    jsonObject.addProperty(columnNames.
                    get(j), currentRow.getCell(j).
                    getNumericCellValue());
                } else if (currentRow.
                getCell(j).getCellTypeEnum() ==
                CellType.BOOLEAN) {
                    jsonObject.addProperty(columnNames.
                    get(j), currentRow.getCell(j).
                    getBooleanCellValue());
                } else if (currentRow.getCell(j).
                getCellTypeEnum() == CellType.BLANK) {
                    jsonObject.addProperty(columnNames.
                    get(j), "");
                }
```

```
                    } else {
                        jsonObject.addProperty(columnNames.
                        get(j), "");
                    }

                }
                saveJSON2(currentRow.getCell(0).
                getStringCellValue(), jsonObject);
            } else if (currentRow.getRowNum() == 0) {
                // store column names
                for (int k = 0; k < currentRow.
                getPhysicalNumberOfCells(); k++) {
                    columnNames.add(currentRow.getCell(k).
                    getStringCellValue());
                }
            }
        }
    }

    public static void saveJSON(String fileName, JsonObject
    jsonObject) throws Exception {
        fileName = JSON_LOCATION + fileName + JSON1_POSTFIX;
        try (Writer writer = new FileWriter(fileName)) {
            Gson gson = new GsonBuilder().setPrettyPrinting().
            serializeNulls().create();
            gson.toJson(jsonObject, writer);
        }
    }

    public static void saveJSON2(String fileName, JsonObject
    jsonObject) throws Exception {
        fileName = JSON_LOCATION + fileName + JSON2_POSTFIX;
        try (Writer writer = new FileWriter(fileName)) {
```

```
            Gson gson = new GsonBuilder().setPrettyPrinting().
            serializeNulls().create();
            gson.toJson(jsonObject, writer);
        }
    }
}
```

I have used two methods in the preceding class that do almost the same functions. The idea is to show that you can alter the logic for a specific Excel file.

Now, let's create another folder, `jsonutil`, in the `src/main/java/com/taf/testautomation/utilities` folder (see screenshot in Figure A-1). Let's create a class `JsonUtil` within this folder, as shown in Listing 9-2. This class will read data from JSON files created from data sources and store these in a map that we can use in the test suite. I have written the code to read data from `source1TestData1.json`. You can enhance this to read test data from multiple sources.

Listing 9-2. JsonUtil.java

```
package com.taf.testautomation.utilities.jsonutil;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.internal.LinkedTreeMap;
import lombok.extern.slf4j.Slf4j;

import java.io.File;
import java.io.FileReader;
import java.io.Reader;
import java.lang.reflect.Type;
import java.util.Map;
```

```
import static com.taf.testautomation.utilities.excelutil.ExcelU
til.getCustomProperties;

@Slf4j
public class JsonUtil {

    private LinkedTreeMap<String, String> customSettings;
    private static final String JSON_LOCATION = "src/test/
    resources/testdata/json";
    private static final String JSON_POSTFIX =
    "TestData1.json";

    public JsonUtil() {
    }

    public Map<String, String> getCustomSettings() {
        String profileName = getProfileFileName(getCustomProper
        ties().get("dataSource1"));
        loadJsonConfig(profileName);
        return customSettings;
    }

    public void loadJsonConfig(String profileName) {
        File resourcesDirectory = new File(JSON_LOCATION);
        File configurationFile = new File(resourcesDirectory,
        profileName);
        try {
            Reader reader = new FileReader(configurationFile);
            GsonBuilder gsonBuilder = new GsonBuilder();
            Gson gson = gsonBuilder.
            enableComplexMapKeySerialization().create();
            customSettings = gson.fromJson(reader, (Type)
            Map.class);
        } catch (Exception e) {
```

```
            JsonUtil.log.error("Error parsing configuration
            file {};", profileName, e);
            throw new RuntimeException(e);
        }
    }

    public String getProfileFileName(String name) {
        return name + JSON_POSTFIX;
    }

}
```

Finally, change the `setUp()` method of the test suite, as shown in Listing 9-3, to import the test data in the test suite.

Listing 9-3. Updated setUp() Method in AboutAppTestSuite.java

```
private AboutAppScreen aboutAppScreen;
private JsonUtil jsonUtil;
private ExcelUtil excelUtil = new ExcelUtil();
protected String testStatus = "";
private static final String SCREEN_NAME = "aboutAppScreen";
private static int i = 0, j = 0;

@BeforeAll
@Override
public void setUp() throws Exception {
    super.setUp();
    if (getCustomProperties().get("loadExcel").equals("Y")) {
        excelUtil.generateJsonFilesFromExcel1();
    }
    jsonUtil = new JsonUtil();
}
```

Now, we will see how we can import data from other file formats.

Importing Test Data from XML and Other Formats

I will now show you the code for XML and txt. You should be able to apply this to any file type.

Let's assume when dealing with Android that we are sourcing test data from XML. And that with iOS, we are sourcing data from a txt file. We will create a sample `data.xml` file within the `src/test/resources/testdata/xml` folder, as shown in Listing 9-4.

Listing 9-4. data.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Test Data for Test Engineer to Architect -->

<resources>
   <!-- Data -->
   <string name="datatype1">Data1</string>
   <string name="datatype2">Data2</string>
   <string name="datatype3">Data3</string>
   <string name="datatype4">Data4</string>
   <string name="datatype5">Data5</string>
   <string name="datatype6">Data6</string>
   <string name="datatype7">Data7</string>
   <string name="datatype8">Data8</string>
   <string name="datatype9">Data9</string>
   <string name="datatype10">Data10</string>

</resources>
```

We will also create a sample `data.txt` file in the `src/test/resources/testdata/txt` folder, as shown in Listing 9-5.

Listing 9-5. data.txt

```
/* Test Data for Test Engineer to Architect */
/* Data */
"datatype1" = "Data1";
"datatype2" = "Data2";
"datatype3" = "Data3";
"datatype4" = "Data4";
"datatype5" = "Data5";
"datatype6" = "Data6";
"datatype7" = "Data7";
"datatype8" = "Data8";
"datatype9" = "Data9";
"datatype10" = "Data10";
```

Now, let's create a folder named `fileutil` within the `src/main/java/com/taf/testautomation/utilities` folder. We will write a `FileUtil` class, as shown in Listing 9-6.

Listing 9-6. FileUtil.java

```
package com.taf.testautomation.utilities.fileutil;

import lombok.extern.slf4j.Slf4j;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import java.io.File;
import java.nio.file.Files;
import java.nio.file.Paths;
```

```
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.stream.Collectors;
import java.util.stream.Stream;

import static com.taf.testautomation.utilities.excelutil.ExcelU
til.getCustomProperties;

@Slf4j
public class FileUtil {

    private LinkedHashMap<String, String> customSettings = new
    LinkedHashMap<>();
    private static final String INPUT_XML_BASE_PATH = "src/
    test/resources/testdata/xml/";
    private static final String INPUT_TXT_BASE_PATH = "src/
    test/resources/testdata/txt/";
    private static final String XML_FILENAME = "data.xml";
    private static final String DATA_FILENAME = "data.txt";

    public FileUtil() {
    }

    public Map<String, String> getCustomSettings() {
        String filePath = getInputFilePath();
        try {
            parseXML(filePath, customSettings);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return customSettings;
    }
```

```
public String getCustomValues(String key) {
    String filePath = getInputFilePath();
    String value = "";
    try {
        value = parseFileLines(filePath, key);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return value;
}

private String getInputFilePath() {
    if (getCustomProperties().get("isAndroid").
    equals("true")) {
        return INPUT_XML_BASE_PATH + XML_FILENAME;
    } else {
        return INPUT_TXT_BASE_PATH + DATA_FILENAME;
    }
}

private void parseXML(String filePath,
LinkedHashMap<String, String> customProperties) throws
Exception {
    File fXmlFile = new File(filePath);
    DocumentBuilderFactory dbFactory = DocumentBuilder
    Factory.newInstance();
    DocumentBuilder dBuilder = dbFactory.
    newDocumentBuilder();
    Document doc = dBuilder.parse(fXmlFile);
    doc.getDocumentElement().normalize();
    log.info("Root element :" + doc.getDocumentElement().
    getNodeName());
```

```
        NodeList nList = doc.getElementsByTagName("string");
        for (int i = 0; i < nList.getLength(); i++) {
            Node nNode = nList.item(i);
            Element eElement = (Element) nNode;
            String key = eElement.getAttribute("name");
            String value = nNode.getTextContent();
            customProperties.put(key, value);
        }
    }

    private String parseFileLines(String filePath,
    String key) {
        String strLine = "";
        try {
            Stream<String> lines = Files.lines(Paths.get
            (filePath));
            strLine = lines.filter(s -> s.contains(key)).
            collect(Collectors.toList()).get(0);
        } catch (Exception e) {
            e.printStackTrace();
        }
        if (getCustomProperties().get("isAndroid").
        equals("true")) {
            return strLine.substring(strLine.indexOf(">") + 1,
            strLine.lastIndexOf("<"));
        } else {
            return strLine.substring(strLine.indexOf("=") + 3,
            strLine.indexOf(";") - 1);
        }
    }
}
```

The method `parseXML(String filePath, LinkedHashMap<String, String> customProperties)` parses the XML files. The parser in the `javax.xml` package considers each XML tag as a node, and then `getAttribute("name")` is used to find the string to be stored as a key in the map. Then, `getTextContent()` captures the value string in the map.

The method `parseFileLines(String filePath, String key)` is more generic and returns the value for a given key. So, it could be used to parse XML, txt, or other file types. In our case, it can be used to read both XML and txt files, which is what it does for Android and iOS, respectively.

Summary

In this chapter, we learned how to read test data from various file types and store that data in maps that we can later use in our test suites. In the next chapter, we will implement BDD with Cucumber and learn how to execute our test suite in BDD mode without any refactoring, meaning we will keep the capability of running the test suite as a Junit test, as we discussed in Chapter 8.

CHAPTER 10

Adding BDD Capabilities with Cucumber

In the previous two chapters, we learned how to write and execute Junit-based test suites. We also discovered how to read test data from various file types. In this chapter, we will add BDD capabilities to our framework with Cucumber so that we can run test suites in either BDD mode or non-BDD mode. BDD is quite popular nowadays as it facilitates teamwork in agile DevOps mode such that product owners can write acceptance criteria in the Gherkins language and the test automation team can take the same acceptance criteria in their feature file. In the next section, we will see how to use a Spring runner class with Cucumber.

Using a Spring Runner Class with Cucumber

To implement BDD capability using Cucumber, we need to create a Cucumber feature file, along with the corresponding step definition file and runner file. Let's call these files `AboutApp`, as shown in Listing 10-1, `AboutAppStepDefinitions`, as shown in Listing 10-2, and `AboutAppTest`, as shown in Listing 10-3. In addition, we also need the `SpringIntegration`

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_10

class, as shown in Listing 10-4, to be able to run Cucumber with Spring. This class configures Cucumber to use the Spring runner class `TestautomationApplication` (Figure 2-3) instead of its own runner class. This is required to run Cucumber with Spring, as our underlying test suites run on the Spring engine.

The code snippets for the preceding feature file and classes are as follows.

Listing 10-1. AboutApp.feature

```
@AboutApp
Feature: AboutApp
  Req number: xxxx

  Background:
    Given application is installed and launched

  @AboutApp @Regression
  Scenario: client wants to verify app screen elements
    When user opens the application
    Then verify create account screen is displayed
    When user fills in details
    And clicks on create account button
    Then verify user is taken to the app login screen
    When user logs in with email and password
    And clicks on sign in button
    Then verified user is taken to the about app screen
    And verified user sees the following in-app screen
      | Screen_Title   |
      | App_Logo       |
      | App_Name       |
      | App_Version    |
      | App_Images     |
    Then close application and update HP QC test run
```

Listing 10-2. AboutAppStepDefinitions.java

```
package com.taf.testautomation.cucumber.aboutapp;

import io.cucumber.datatable.DataTable;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;

import java.util.List;

public class AboutAppStepDefinitions extends
SpringIntegration {

    @Given("application is installed and launched")
    public void application_is_installed_and_launched() throws
    Exception {
        setUp();
    }

    @When("user opens the application")
    public void user_opens_the_application() {
        navigateToScreen();
    }

    @Then("verify create account screen is displayed")
    public void verify_create_account_screen_is_displayed() {
        log("leaving the implementation to reader");
    }

    @When("user fills in details")
    public void user_fills_in_details() {
        log("this step is covered in navigateToScreen()");
    }
```

```
@And("clicks on create account button")
public void clicks_on_create_account_button() {
    log("this step is covered in navigateToScreen()");
}

@Then("verify user is taken to the app login screen")
public void verify_user_is_taken_to_the_app_login_screen() {
    log("leaving the implementation to reader");
}

@When("user logs in with email and password")
public void user_logs_in_with_email_and_password() {
    log("this step is covered in navigateToScreen()");
}

@And("clicks on sign in button")
public void clicks_on_sign_in_button() {
    log("this step is covered in navigateToScreen()");
}

@Then("verified user is taken to the about app screen")
public void verify_user_is_taken_to_the_about_app_screen() {
    log("leaving the implementation to reader");
}

@And("verified user sees the following in-app screen")
public void verify_user_sees_the_following_in_app_
screen(DataTable dt) {
    List<String> list = dt.asList(String.class);
    dataTable = new String[list.size()][1];
    for (int i = 0; i < list.size(); i++) {
        dataTable[i][0] = list.get(i);
    }
```

```
        for (String str : list) {
            switch (str) {
                case "Screen_Title":
                    testScenario1();
                    break;
                case "App_Logo":
                    testScenario2();
                    break;
                case "App_Name":
                    testScenario3();
                    break;
                case "App_Version":
                    testScenario4();
                    break;
                default:
                    testScenario5();
                    break;
            }
        }
    }

    @Then("close application and update HP QC test run")
    public void close_application_and_update_HP_QC_test_run()
    throws Exception {
        tearDown();
    }

}
```

Listing 10-3. AboutAppTest.java

```
package com.taf.testautomation.cucumber.aboutapp;

import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(features = "src/test/resources/features", glue
= {"com.taf.testautomation.cucumber.aboutapp"}, monochrome =
true, plugin = {"html:build/cucumber-html-report-normal",
        "json:build/cucumber.json", "com.taf.testautomation.
cucumber.ExtentCucumberAdapter:"}, tags = {"@AboutApp"})
public class AboutAppTest {

}
```

Listing 10-4. SpringIntegration.java

```
package com.taf.testautomation.cucumber.aboutapp;

import com.taf.testautomation.TestautomationApplication;
import com.taf.testautomation.uitests.AboutAppTestSuite;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit.jupiter.
SpringExtension;

@ExtendWith(SpringExtension.class)
@SpringBootTest(classes = {TestautomationApplication.class})
```

```
@ContextConfiguration(classes =
TestautomationApplication.class)
public class SpringIntegration extends AboutAppTestSuite {

}
```

The AboutApp feature file (Listing 10-1) captures the screen navigation options for our hypothetical app. I have used a datable to list the verification points. The AboutAppStepDefinitions class (Listing 10-2) defines these steps. Here, since my data table doesn't have a header, I capture its content in a list of String data type. You may use a map in cases where a datable has a header. You will notice that I am assigning the value of the static variable dataTable that we defined in the BaseTest class. I will use this variable to print the data table in the extent report (Chapter 11).

Please note the way we use tags in Cucumber feature file. The same tag (in our case, @AboutApp) needs to be specified in the runner file so that the runner knows which scenarios to execute.

Generating Extent Report in Runner Class

The AboutAppTest runner file (Listing 10-3) uses a custom Extent adaptor, which we will create in Chapter 11. This is done because Junit does not have a default extent listener for Cucumber. Now, let's discuss the step definitions.

Writing Step Definitions

In this book, we are creating a framework that supports both TDD and BDD. To do this, in the BDD step definition file we basically call the test methods in AboutAppTestSuite. Thus, our step definition file needs to extend the SpringIntegration class, which then needs to extend the

`AboutAppTestSuite` class. This is how we integrate Cucumber with any TDD framework without dismantling the existing code base. Now, it is time to run our test suite!

Running Test Suite in Gradle

To run the BDD runner file, use the following command. This will also generate the extent report at the defined location (see Chapter 11):

```
./gradlew test -Dcucumber.options="--tags @AboutApp"
```

Summary

In this chapter, we learned how to implement BDD with Cucumber as a wrapper over our Junit tests. We also learned how to execute our BDD test. In the next chapter, we will learn how to add an Allure report and an Extent report for our tests.

CHAPTER 11

Adding Allure and Enhanced Extent Reports

In the previous chapter, we learned how to write and execute Cucumber-based test suites. In this chapter, we will learn how to generate two important and widely used reports, namely, Allure and Extent reports. First, we will see how to generate an Allure report with Junit and Cucumber.

Generating Allure Report

Let's create a property file called `allure.properties` in the `src/test/resources` folder, as shown in Listing 11-1. This is where the generated Allure report will be stored.

Listing 11-1. allure.properties

```
allure.results.directory=build/allure-results
```


K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_11

Please recall that when discussing the creation of page objects (Chapter 7) and test suites (Chapter 8) we mentioned Allure annotations. These are what appear in the Allure report. You can order the content in the Allure report by Epic or Feature type, which makes the Allure report very appealing.

To generate an Allure report after running the test suite, use the following command with the appropriate Gradle task. To run test methods tagged with Smoke, use

```
./gradlew Smoke --tests "com.taf.testautomation.uitests.
AboutAppTestSuite" allurereport
```

If you want to generate an Allure report for Cucumber, add the `AllureCucumber5Jvm` plugin to the runner file, as shown in Listing 11-2. We do this because we are using the `allure-cucumber5-jvm` plugin in build.gradle.

Listing 11-2. Allure Plugin in AboutAppTest Runner File

```
@CucumberOptions(features = "src/test/resources/features", glue
= {"com.taf.testautomation.cucumber.aboutapp"}, monochrome =
true, plugin = {"html:build/cucumber-html-report-normal",
        "json:build/cucumber.json", "io.qameta.allure.
cucumber5jvm.AllureCucumber5Jvm", "com.taf.testautomation.
cucumber.ExtentCucumberAdapter:"}, tags = {"@AboutApp"})
```

Then, from the Gradle command line, run the runner file, as follows:

```
./gradlew test -Dcucumber.options="--tags @AboutApp"
allurereport
```

Viewing Allure Report

The Allure report will be generated in the Gradle build folder. You can either manually open the Allure report in a browser or use the following command to open it in your default browser after the preceding command is run:

```
allure serve build/allure-results
```

Now we will see how to generate an Extent report with BDD.

Making Extent Report Work with Junit

Similar to the Allure report, we will need an `extent.properties` file, as shown in Listing 11-3, to create an Extent report. We will also need the file `avent-config.xml`, as shown in Listing 11-4, to define the Extent report format. I have created these files within the `src/test/resources` folder, as shown in Listing 11-3.

Listing 11-3. extent.properties

```
extent.reporter.avent.start=false
extent.reporter.bdd.start=false
extent.reporter.cards.start=false
extent.reporter.email.start=false
extent.reporter.html.start=true
extent.reporter.klov.start=false
extent.reporter.logger.start=false
extent.reporter.tabular.start=false

extent.reporter.avent.out=test-result/AventReport/
extent.reporter.bdd.out=test-result/BddReport/
extent.reporter.cards.out=test-result/CardsReport/
extent.reporter.email.out=test-result/emailreport/
ExtentEmail.html
```

```
extent.reporter.html.out=test-result/htmlreport/ExtentHtml.html
extent.reporter.logger.out=test-result/LoggerReport/
extent.reporter.tabular.out=test-result/TabularReport/
```

Listing 11-4. avent-config.xml

```
Avent-config.xml:
<?xml version="1.0" encoding="UTF-8"?>
<extentreports>
    <configuration>
        <!-- report theme -->
        <!-- standard, dark -->
        <theme>standard</theme>

        <!-- viewstyle -->
        <!-- alt-view -->
        <viewstyle></viewstyle>

        <!-- document encoding -->
        <!-- defaults to UTF-8 -->
        <encoding>UTF-8</encoding>

        <!-- offline report -->
        <enableOfflineMode>false</enableOfflineMode>

        <!-- enable or disable timeline on dashboard -->
        <enableTimeline>true</enableTimeline>

        <!-- protocol for script and stylesheets -->
        <!-- defaults to https -->
        <protocol>https</protocol>

        <!-- title of the document -->
        <documentTitle>Extent Framework</documentTitle>
```

```
        <!-- report name - displayed at top-nav -->
        <reportName>Build 1</reportName>

        <!-- timestamp format -->
        <timeStampFormat>MMM dd, yyyy HH:mm:ss</
        timeStampFormat>

        <!-- custom javascript -->
        <scripts>
            <![CDATA[
                $(document).ready(function() {

                });
            ]]>
        </scripts>

        <!-- custom styles -->
        <styles>
            <![CDATA[

            ]]>
        </styles>
    </configuration>
</extentreports>
```

In `extent.properties`, I have enabled the HTML report and specified the HTML report location. I created the folder `test-result/htmlreport` under the project root. I also created two other folders, namely `pdfreport` and `screenshots`, within the `test-result` folder. We will use these in the next two chapters. So, now our folder structure looks like that in Figure 11-1.

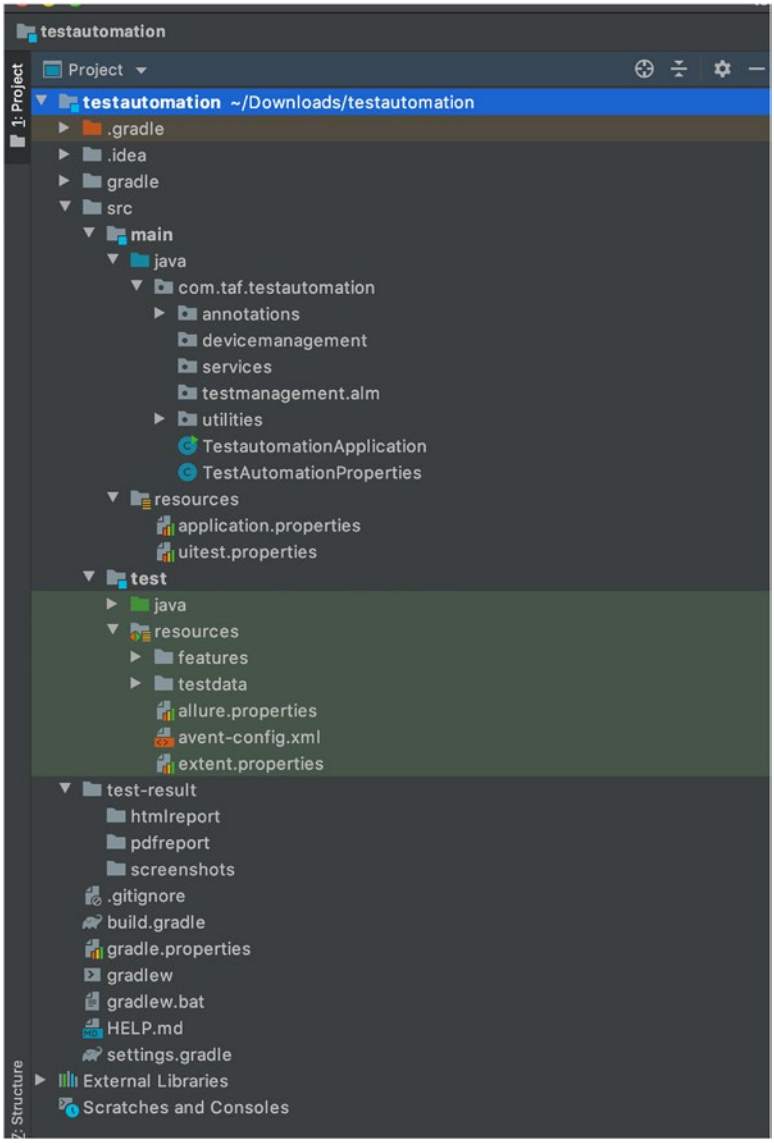


Figure 11-1. *Folder structure for reports and screenshots*

As we mentioned in Chapter 10 and elsewhere, we have to customize the default Extent adaptor to make it work with Junit. For this, I have used the Extent adaptor available at https://github.com/extent-framework/extentreports-cucumber4-adapter and updated it to suit our needs. The classes that we need in order to implement the custom adaptors are ExtentCucumberAdapter, TestSourcesModel, and URLOutputStream, as shown in Listings 11-5, 11-6, and 11-7. We will make ExtentCucumberAdapter extend our BaseTest (Listing 8-1) so that we can print the data table in the generated Extent report. The code snippets for the classes are as follows.

Listing 11-5. ExtentCucumberAdapter.java

```
package com.taf.testautomation.cucumber;

import com.taf.testautomation.BaseTest;
import com.aventstack.extentreports.ExtentTest;
import com.aventstack.extentreports.GherkinKeyword;
import com.aventstack.extentreports.MediaEntityBuilder;
import com.aventstack.extentreports.Status;
import com.aventstack.extentreports.gherkin.model.Asterisk;
import com.aventstack.extentreports.markuputils.MarkupHelper;
import com.aventstack.extentreports.model.service.LogService;
import com.aventstack.extentreports.model.service.TestService;
import com.aventstack.extentreports.reporter.
ExtentHtmlReporter;
import com.aventstack.extentreports.service.ExtentService;
import io.cucumber.core.exception.CucumberException;
import io.cucumber.core.gherkin.vintage.internal.gherkin.
pickles.PickleCell;
import io.cucumber.core.gherkin.vintage.internal.gherkin.
pickles.PickleRow;
import io.cucumber.core.gherkin.vintage.internal.gherkin.
pickles.PickleString;
```

```
import io.cucumber.core.gherkin.vintage.internal.gherkin.
pickles.PickleTable;
import io.cucumber.core.internal.gherkin.ast.*;
import io.cucumber.plugin.ConcurrentEventListener;
import io.cucumber.plugin.event.*;
import lombok.SneakyThrows;
import lombok.extern.slf4j.Slf4j;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.OutputStream;
import java.net.URISyntaxException;
import java.net.URL;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.stream.Collectors;

/**
 * A port of Cucumber-JVM (MIT licensed) HtmlFormatter for
Extent Framework
 * Original source: https://github.com/cucumber/cucumber-jvm/
blob/master/core/src/main/java/cucumber/runtime/formatter/
HTMLFormatter.java
 */
@Slf4j
public class ExtentCucumberAdapter extends BaseTest
        implements ConcurrentEventListener {
```

```
private static final String SCREENSHOT_DIR_PROPERTY =
"screenshot.dir";
private static final String SCREENSHOT_REL_PATH_PROPERTY =
"screenshot.rel.path";

private static Map<String, ExtentTest> featureMap = new
ConcurrentHashMap<>();
private static ThreadLocal<ExtentTest>
featureTestThreadLocal = new InheritableThreadLocal<>();
private static Map<String, ExtentTest> scenarioOutlineMap =
new ConcurrentHashMap<>();
private static ThreadLocal<ExtentTest>
scenarioOutlineThreadLocal = new
InheritableThreadLocal<>();
private static ThreadLocal<ExtentTest> scenarioThreadLocal
= new InheritableThreadLocal<>();
private static ThreadLocal<Boolean> isHookThreadLocal = new
InheritableThreadLocal<>();
private static ThreadLocal<ExtentTest> stepTestThreadLocal
= new InheritableThreadLocal<>();

private String screenshotDir;
private String screenshotRelPath;

@SuppressWarnings("serial")
private static final Map<String, String> MIME_TYPES_
EXTENSIONS = new HashMap<String, String>() {
    {
        put("image/bmp", "bmp");
        put("image/gif", "gif");
        put("image/jpeg", "jpg");
        put("image/png", "png");
```

```
            put("image/svg+xml", "svg");
            put("video/ogg", "ogg");
        }
    };

    private static final AtomicInteger EMBEDDED_INT = new
    AtomicInteger(0);
    private static final String REPORT_PATH = "test-result/
    htmlreport/ExtentHtml.html";
    private static final String REPORT_CONFIG = "src/test/
    resources/avent-config.xml";

    private final TestSourcesModel testSources = new
    TestSourcesModel();

    private ThreadLocal<String> currentFeatureFile = new
    ThreadLocal<>();
    private ThreadLocal<ScenarioOutline> currentScenarioOutline
    = new InheritableThreadLocal<>();
    private ThreadLocal<Examples> currentExamples = new
    InheritableThreadLocal<>();

    private EventHandler<TestSourceRead> testSourceReadHandler
    = new EventHandler<TestSourceRead>() {
        @Override
        public void receive(TestSourceRead event) {
            handleTestSourceRead(event);
        }
    };
   private EventHandler<TestCaseStarted> caseStartedHandler =
   new EventHandler<TestCaseStarted>() {
        @SneakyThrows
```

```
    @Override
    public void receive(TestCaseStarted event) {
        handleTestCaseStarted(event);
    }
};
private EventHandler<TestStepStarted> stepStartedHandler =
new EventHandler<TestStepStarted>() {
    @SneakyThrows
    @Override
    public void receive(TestStepStarted event) {
        handleTestStepStarted(event);
    }
};
private EventHandler<TestStepFinished> stepFinishedHandler
= new EventHandler<TestStepFinished>() {
    @Override
    public void receive(TestStepFinished event) {
        handleTestStepFinished(event);
    }
};
private EventHandler<EmbedEvent> embedEventhandler = new
EventHandler<EmbedEvent>() {
    @Override
    public void receive(EmbedEvent event) {
        handleEmbed(event);
    }
};
private EventHandler<WriteEvent> writeEventhandler = new
EventHandler<WriteEvent>() {
    @Override
    public void receive(WriteEvent event) {
```

```
            handleWrite(event);
        }
    };
    private EventHandler<TestRunFinished> runFinishedHandler =
    new EventHandler<TestRunFinished>() {
        @Override
        public void receive(TestRunFinished event) {
            finishReport();
        }
    };

    public ExtentCucumberAdapter(String arg) {
        ExtentService.getInstance();
        Object prop = ExtentService.getProperty(SCREENSHOT_DIR_
        PROPERTY);
        screenshotDir = prop == null ? "test-output/" :
        String.valueOf(prop);
        prop = ExtentService.getProperty(SCREENSHOT_REL_PATH_
        PROPERTY);
        screenshotRelPath = prop == null ||
        String.valueOf(prop).isEmpty() ? screenshotDir :
        String.valueOf(prop);
        screenshotRelPath = screenshotRelPath == null ? "" :
        screenshotRelPath;
    }

    @Override
    public void setEventPublisher(EventPublisher publisher) {
        publisher.registerHandlerFor(TestSourceRead.class,
        testSourceReadHandler);
        publisher.registerHandlerFor(TestCaseStarted.class,
        caseStartedHandler);
```

```
        publisher.registerHandlerFor(TestStepStarted.class,
        stepStartedHandler);
        publisher.registerHandlerFor(TestStepFinished.class,
        stepFinishedHandler);
        publisher.registerHandlerFor(EmbedEvent.class,
        embedEventhandler);
        publisher.registerHandlerFor(WriteEvent.class,
        writeEventhandler);
        publisher.registerHandlerFor(TestRunFinished.class,
        runFinishedHandler);
    }

    private void handleTestSourceRead(TestSourceRead event) {
        testSources.addTestSourceReadEvent(event.getUri().
        toString(), event);
    }

    private synchronized void handleTestCaseStarted(TestCaseSta
    rted event) throws FileNotFoundException {
        handleStartOfFeature(event.getTestCase());
        handleScenarioOutline(event.getTestCase());
        createTestCase(event.getTestCase());
        if (testSources.hasBackground(currentFeatureFile.get(),
        event.getTestCase().getLine())) {
            // background
        }
    }

    private synchronized void handleTestStepStarted(TestStep
    Started event) {
        isHookThreadLocal.set(false);
```

```
        log.info("Test step is" + event.getTestStep().
        toString());

        if (event.getTestStep() instanceof HookTestStep) {
            ExtentTest t = scenarioThreadLocal.get()
                    .createNode(Asterisk.class, event.
                    getTestStep().getCodeLocation());
            stepTestThreadLocal.set(t);
            isHookThreadLocal.set(true);
        }

        if (event.getTestStep() instanceof
        PickleStepTestStep) {
            PickleStepTestStep testStep = (PickleStepTestStep)
            event.getTestStep();
            createTestStep(testStep);
        }
    }

    private synchronized void handleTestStepFinished(TestStepFi
    nished event) {
        updateResult(event.getResult());
    }

    private synchronized void updateResult(Result result) {
        switch (result.getStatus().toString()) {
            case "FAILED":
                stepTestThreadLocal.get().fail(result.
                getError());
                break;
            case "SKIPPED":
            case "PENDING":
```

```
        Boolean currentEndingEventSkipped =
        stepTestThreadLocal.get().getModel().
        getLogContext() != null
                && !stepTestThreadLocal.get().
                getModel().getLogContext().isEmpty()
                ? stepTestThreadLocal.get().getModel().
                getLogContext().getLast().getStatus()
                == Status.SKIP
                : false;
        if (result.getError() != null) {
            stepTestThreadLocal.get().skip(result.
            getError());
        } else if (!currentEndingEventSkipped) {
            String details = result.toString() == null
            ? "Step skipped" : result.toString();
            stepTestThreadLocal.get().skip(details);
        }
        break;
    case "PASSED":
        if (stepTestThreadLocal.get() !=
        null && stepTestThreadLocal.get().
        getModel().getLogContext().isEmpty() &&
        !isHookThreadLocal.get()) {
            ExtentHtmlReporter avent = new
            ExtentHtmlReporter(REPORT_PATH);
            ExtentService.getInstance().
            attachReporter(avent);
            avent.loadXMLConfig(REPORT_CONFIG);
            stepTestThreadLocal.get().pass("");
        }
        if (stepTestThreadLocal.get() != null) {
```

```
                    Boolean hasLog = TestService.testHasLog
                    (stepTestThreadLocal.get().getModel());
                    Boolean hasScreenCapture = hasLog && LogS
                    ervice.logHasScreenCapture(stepTestThrea
                    dLocal.get().getModel().getLogContext().
                    getFirst());
                    if (isHookThreadLocal.get() && !hasLog &&
                    !hasScreenCapture) {
                        ExtentService.getInstance().removeTest(
                        stepTestThreadLocal.get());
                    }
                }
                break;
            default:
                break;
        }
    }

    private synchronized void handleEmbed(EmbedEvent event) {
        String mimeType = event.getMediaType();
        String extension = MIME_TYPES_EXTENSIONS.get(mimeType);
        if (extension != null) {
            StringBuilder fileName = new
            StringBuilder("embedded").append(EMBEDDED_INT.
            incrementAndGet()).append(".").append(extension);
            try {
                URL url = toUrl(fileName.toString());
                writeBytesToURL(event.getData(), url);
                try {
                    File f = new File(url.toURI());
                    if (stepTestThreadLocal.get() == null) {
                        ExtentTest t = scenarioThreadLocal.get()
```

```
                        .createNode(Asterisk.class,
                        "Embed");
                    stepTestThreadLocal.set(t);
                }
                stepTestThreadLocal.get().info("", MediaEnt
                ityBuilder.createScreenCaptureFromPath
                (screenshotRelPath + f.getName()).build());
            } catch (URISyntaxException e) {
                e.printStackTrace();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

private static void writeBytesToURL(byte[] buf, URL url)
throws IOException {
    OutputStream out = createReportFileOutputStream(url);
    try {
        out.write(buf);
    } catch (IOException e) {
        throw new IOException("Unable to write to report
        file item: ", e);
    }
}

private static OutputStream createReportFileOutputStream
(URL url) {
    try {
        return new URLOutputStream(url);
    } catch (IOException e) {
```

```
            throw new CucumberException(e);
        }
    }

    private URL toUrl(String fileName) {
        try {
            URL url = Paths.get(screenshotDir, fileName).
            toUri().toURL();
            return url;
        } catch (IOException e) {
            throw new CucumberException(e);
        }
    }

    private void handleWrite(WriteEvent event) {
        String text = event.getText();
        if (text != null && !text.isEmpty()) {
            stepTestThreadLocal.get().info(text);
        }
    }

    private void finishReport() {
        ExtentService.getInstance().flush();
    }

    private synchronized void handleStartOfFeature(TestCase
    testCase) {
        if (currentFeatureFile == null || !currentFeatureFile.
        equals(testCase.getUri())) {
            currentFeatureFile.set(testCase.getUri().
            toString());
            createFeature(testCase);
        }
    }
```

```
private synchronized void createFeature(TestCase testCase) {
    Feature feature = testSources.getFeature(testCase.
    getUri().toString());
    if (feature != null) {
        if (featureMap.containsKey(feature.getName())) {
            featureTestThreadLocal.set(featureMap.
            get(feature.getName()));
            return;
        }
        if (featureTestThreadLocal.get() != null &&
        featureTestThreadLocal.get().getModel().getName().
        equals(feature.getName())) {
            return;
        }
        ExtentTest t = ExtentService.getInstance()
                .createTest(com.aventstack.extentreports.
                gherkin.model.Feature.class, feature.
                getName(), feature.getDescription());
        featureTestThreadLocal.set(t);
        featureMap.put(feature.getName(), t);
        List<String> tagList = createTagsList(feature.
        getTags());
        tagList.forEach(featureTestThreadLocal.
        get()::assignCategory);
    }
}

private List<String> createTagsList(List<Tag> tags) {
    List<String> tagList = new ArrayList<>();
    for (Tag tag : tags) {
        tagList.add(tag.getName());
    }
```

```
        return tagList;
    }

    private synchronized void handleScenarioOutline(TestCase
    testCase) {
        TestSourcesModel.AstNode astNode = testSources.getAstNode
        (currentFeatureFile.get(), testCase.getLine());
        if (TestSourcesModel.isScenarioOutlineScenario
        (astNode)) {
            ScenarioOutline scenarioOutline = (ScenarioOutline)
            TestSourcesModel.getScenarioDefinition(astNode);
            if (currentScenarioOutline.get() == null ||
            !currentScenarioOutline.get().getName().
            equals(scenarioOutline.getName())) {
                scenarioOutlineThreadLocal.set(null);
                createScenarioOutline(scenarioOutline);
                currentScenarioOutline.set(scenarioOutline);
                addOutlineStepsToReport(scenarioOutline);
            }
            Examples examples = (Examples) astNode.parent.node;
            if (currentExamples.get() == null ||
            !currentExamples.get().equals(examples)) {
                currentExamples.set(examples);
                createExamples(examples);
            }
        } else {
            scenarioOutlineThreadLocal.set(null);
            currentScenarioOutline.set(null);
            currentExamples.set(null);
        }
    }
```

```
private synchronized void createScenarioOutline(ScenarioOut
line scenarioOutline) {
    if (scenarioOutlineMap.containsKey(scenarioOutline.
    getName())) {
        scenarioOutlineThreadLocal.set(scenarioOutlineMap.
        get(scenarioOutline.getName()));
        return;
    }
    if (scenarioOutlineThreadLocal.get() == null) {
        ExtentTest t = featureTestThreadLocal.get()
                .createNode(com.aventstack.extentreports.
                gherkin.model.ScenarioOutline.class,
                scenarioOutline.getName(), scenarioOutline.
                getDescription());
        scenarioOutlineThreadLocal.set(t);
        scenarioOutlineMap.put(scenarioOutline.
        getName(), t);
        List<String> featureTags = scenarioOutlineThreadLocal
        .get().getModel()
                .getParent().getCategoryContext().getAll()
                .stream()
                .map(x -> x.getName())
                .collect(Collectors.toList());
        scenarioOutline.getTags()
                .stream()
                .map(x -> x.getName())
                .filter(x -> !featureTags.contains(x))
                .forEach(scenarioOutlineThreadLocal.
                get()::assignCategory);
    }
}
```

```
    private synchronized void addOutlineStepsToReport(ScenarioO
    utline scenarioOutline) {
        for (io.cucumber.core.internal.gherkin.ast.Step step :
        scenarioOutline.getSteps()) {
            if (step.getArgument() != null) {
                Node argument = step.getArgument();
                if (argument instanceof DocString) {
                    createDocStringMap((DocString) argument);
                } else if (argument instanceof DataTable) {

                }
            }
        }
    }

    private Map<String, Object> createDocStringMap(DocString
    docString) {
        Map<String, Object> docStringMap = new HashMap<String,
        Object>();
        docStringMap.put("value", docString.getContent());
        return docStringMap;
    }

    private void createExamples(Examples examples) {
        List<TableRow> rows = new ArrayList<>();
        rows.add(examples.getTableHeader());
        rows.addAll(examples.getTableBody());
        String[][] data = getTable(rows);
        String markup = MarkupHelper.createTable(data).
        getMarkup();
        if (examples.getName() != null && !examples.getName().
        isEmpty()) {
            markup = examples.getName() + markup;
        }
```

```
        markup = scenarioOutlineThreadLocal.get().getModel().
        getDescription() + markup;
        scenarioOutlineThreadLocal.get().getModel().
        setDescription(markup);
    }

    private String[][] getTable(List<TableRow> rows) {
        String data[][] = null;
        int rowSize = rows.size();
        for (int i = 0; i < rowSize; i++) {
            TableRow row = rows.get(i);
            List<TableCell> cells = row.getCells();
            int cellSize = cells.size();
            if (data == null) {
                data = new String[rowSize][cellSize];
            }
            for (int j = 0; j < cellSize; j++) {
                data[i][j] = cells.get(j).getValue();
            }
        }
        return data;
    }

    private synchronized void createTestCase(TestCase
    testCase) {
        TestSourcesModel.AstNode astNode = testSources.getAstNode
        (currentFeatureFile.get(), testCase.getLine());
        if (astNode != null) {
            ScenarioDefinition scenarioDefinition = TestSources
            Model.getScenarioDefinition(astNode);
            ExtentTest parent = scenarioOutlineThreadLocal.
            get() != null ? scenarioOutlineThreadLocal.get() :
            featureTestThreadLocal.get();
```

```
            ExtentTest t = parent.createNode(com.aventstack.
            extentreports.gherkin.model.Scenario.class,
            scenarioDefinition.getName(), scenarioDefinition.
            getDescription());
            scenarioThreadLocal.set(t);
        }
        if (!testCase.getTags().isEmpty()) {
            testCase.getTags()
                    .stream()
                    .forEach(scenarioThreadLocal.get()
                     ::assignCategory);
        }
    }

    private synchronized void createTestStep(PickleStepTestStep
    testStep) {
        String stepName = testStep.getStep().getText();
        TestSourcesModel.AstNode astNode = testSources.
        getAstNode(currentFeatureFile.get(), testStep.
        getStep().getLine());
        if (astNode != null) {
            io.cucumber.core.internal.gherkin.ast.Step step
            = (io.cucumber.core.internal.gherkin.ast.Step)
            astNode.node;
            String str = step.getKeyword();
            log.info("Keyword value: " + str);
            try {
                String name = stepName == null || stepName.
                isEmpty()
                        ? step.getText().replace("<", "&lt;").
                        replace(">", "&gt;")
                        : stepName;
```

```
            ExtentTest t = scenarioThreadLocal.get()
                    .createNode(new GherkinKeyword(str.
                    trim()), str + name, testStep.
                    getCodeLocation());
            stepTestThreadLocal.set(t);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
    if (!testStep.getStep().getText().isEmpty()) {
        log.info("test step: " + testStep.getStep().
        getText());
        StepArgument argument = testStep.getStep().
        getArgument();
        if (argument instanceof PickleString) {
            createDocStringMap((PickleString) argument);
        } else if (argument instanceof PickleTable) {
            List<PickleRow> rows = ((PickleTable) argument)
            .getRows();
            stepTestThreadLocal.get().pass(MarkupHelper.cre
            ateTable(getPickleTable(rows)).getMarkup());
        }
        if (testStep.getStep().getText().contains
        ("following")) {
            stepTestThreadLocal.get().pass(MarkupHelper.cre
            ateTable(dataTable).getMarkup());
        }
    }
}

private String[][] getPickleTable(List<PickleRow> rows) {
    String data[][] = null;
```

```
        int rowSize = rows.size();
        for (int i = 0; i < rowSize; i++) {
            PickleRow row = rows.get(i);
            List<PickleCell> cells = row.getCells();
            int cellSize = cells.size();
            if (data == null) {
                data = new String[rowSize][cellSize];
            }
            for (int j = 0; j < cellSize; j++) {
                data[i][j] = cells.get(j).getValue();
            }
        }
        return data;
    }

    private Map<String, Object> createDocStringMap(PickleString
    docString) {
        Map<String, Object> docStringMap = new HashMap<String,
        Object>();
        docStringMap.put("value", docString.getContent());
        return docStringMap;
    }

    // the below additions are from PR #33
    // https://github.com/extent-framework/extentreports-
    cucumber4-adapter/pull/33
    public static synchronized void addTestStepLog(String
    message) {
        stepTestThreadLocal.get().info(message);
    }

    public static synchronized void addTestStepScreenCapture
    FromPath(String imagePath) throws IOException {
```

```
        stepTestThreadLocal.get().addScreenCaptureFromPath(i
        magePath);
    }

    public static synchronized void addTestStepScreenCapture
    FromPath(String imagePath, String title) throws IOException {
        stepTestThreadLocal.get().addScreenCaptureFromPath
        (imagePath, title);
    }

    public static ExtentTest getCurrentStep() {
        return stepTestThreadLocal.get();
    }
}
```

Listing 11-6. TestSourcesModel.java

```
package com.taf.testautomation.cucumber;

import io.cucumber.core.internal.gherkin.AstBuilder;
import io.cucumber.core.internal.gherkin.Parser;
import io.cucumber.core.internal.gherkin.ParserException;
import io.cucumber.core.internal.gherkin.TokenMatcher;
import io.cucumber.core.internal.gherkin.ast.*;
import io.cucumber.plugin.event.TestSourceRead;
import lombok.extern.slf4j.Slf4j;

import java.util.HashMap;
import java.util.Map;

@Slf4j
final class TestSourcesModel {
    private final Map<String, TestSourceRead>
    pathToReadEventMap = new HashMap<>();
```

```
    private final Map<String, GherkinDocument> pathToAstMap =
    new HashMap<String, GherkinDocument>();
    private final Map<String, Map<Integer, AstNode>>
    pathToNodeMap = new HashMap<String, Map<Integer,
    AstNode>>();

    private static Feature getFeatureForTestCase(AstNode
    astNode) {
        while (astNode.parent != null) {
            astNode = astNode.parent;
        }
        return (Feature) astNode.node;
    }

    static Background getBackgroundForTestCase(AstNode
    astNode) {
        Feature feature = getFeatureForTestCase(astNode);
        ScenarioDefinition backgound = feature.
        getChildren().get(0);
        if (backgound instanceof Background) {
            return (Background) backgound;
        } else {
            return null;
        }
    }

        static ScenarioDefinition getScenarioDefinition(AstNode
        astNode) {
        return astNode.node instanceof ScenarioDefinition
        ? (ScenarioDefinition) astNode.node :
        (ScenarioDefinition) astNode.parent.parent.node;
    }

    static boolean isScenarioOutlineScenario(AstNode astNode) {
```

```
    return !(astNode.node instanceof ScenarioDefinition);
}

static boolean isBackgroundStep(AstNode astNode) {
    return astNode.parent.node instanceof Background;
}

static String calculateId(AstNode astNode) {
    Node node = astNode.node;
    if (node instanceof ScenarioDefinition) {
        return calculateId(astNode.parent) + ";" + convertT
        oId(((ScenarioDefinition) node).getName());
    }
    if (node instanceof ExamplesRowWrapperNode) {
        return calculateId(astNode.parent) + ";" +
        (((ExamplesRowWrapperNode) node).bodyRowIndex + 2);
    }
    if (node instanceof TableRow) {
        return calculateId(astNode.parent) + ";" + 1;
    }
    if (node instanceof Examples) {
        return calculateId(astNode.parent) + ";" +
        convertToId(((Examples) node).getName());
    }
    if (node instanceof Feature) {
        return convertToId(((Feature) node).getName());
    }
    return "";
}

static String convertToId(String name) {
    return name.replaceAll("[\\s'_,!]", "-").toLowerCase();
}
```

```
    void addTestSourceReadEvent(String path, TestSourceRead
    event) {
        pathToReadEventMap.put(path, event);
    }

    Feature getFeature(String path) {
        if (!pathToAstMap.containsKey(path)) {
            parseGherkinSource(path);
        }
        if (pathToAstMap.containsKey(path)) {
            return pathToAstMap.get(path).getFeature();
        }
        return null;
    }

    ScenarioDefinition getScenarioDefinition(String path,
    int line) {
        return getScenarioDefinition(getAstNode(path, line));
    }

    AstNode getAstNode(String path, int line) {
        if (!pathToNodeMap.containsKey(path)) {
            parseGherkinSource(path);
        }
        if (pathToNodeMap.containsKey(path)) {
            return pathToNodeMap.get(path).get(line);
        }
        return null;
    }

    boolean hasBackground(String path, int line) {
        if (!pathToNodeMap.containsKey(path)) {
            parseGherkinSource(path);
```

```
    }
    if (pathToNodeMap.containsKey(path)) {
        AstNode astNode = pathToNodeMap.get(path).
        get(line);
        return getBackgroundForTestCase(astNode) != null;
    }
    return false;
}

private TestSourceRead getTestSourceReadEvent(String uri) {
    if (pathToReadEventMap.containsKey(uri)) {
        return pathToReadEventMap.get(uri);
    }
    return null;
}

String getFeatureName(String uri) {
    Feature feature = getFeature(uri);
    if (feature != null) {
        return feature.getName();
    }
    return "";
}

private void parseGherkinSource(String path) {
    if (!pathToReadEventMap.containsKey(path)) {
        return;
    }
    log.info("Gherkin File:"+pathToReadEventMap.get(path).
    getSource());
    Parser<GherkinDocument> parser = new
    Parser<GherkinDocument>(new AstBuilder());
    TokenMatcher matcher = new TokenMatcher();
```

```
        try {
            GherkinDocument gherkinDocument = parser.
            parse(pathToReadEventMap.get(path).getSource(),
            matcher);
            pathToAstMap.put(path, gherkinDocument);
            log.info("Feature file path "+ path);
            Map<Integer, AstNode> nodeMap = new HashMap<>();
            AstNode currentParent = new
            AstNode(gherkinDocument.getFeature(), null);
            log.info("Gherkin Document "+gherkinDocument.
            getFeature());
            for (ScenarioDefinition child : gherkinDocument.
            getFeature().getChildren()) {
                processScenarioDefinition(nodeMap, child,
                currentParent);
            }
            pathToNodeMap.put(path, nodeMap);
            log.info("ScenarioDefinition Steps:"+pathToNodeMap.
            get(path));
        } catch (ParserException e) {
            // Ignore exceptions
        }
    }

    private void processScenarioDefinition(Map<Integer,
    AstNode> nodeMap, ScenarioDefinition child, AstNode
    currentParent) {
        AstNode childNode = new AstNode(child, currentParent);
        nodeMap.put(child.getLocation().getLine(), childNode);
        for (Step step : child.getSteps()) {
            nodeMap.put(step.getLocation().getLine(), new
            AstNode(step, childNode));
```

```
    }
    if (child instanceof ScenarioOutline) {
        processScenarioOutlineExamples(nodeMap,
        (ScenarioOutline) child, childNode);
    }
}

private void processScenarioOutlineExamples(Map<Integer,
AstNode> nodeMap, ScenarioOutline scenarioOutline, AstNode
childNode) {
    for (Examples examples : scenarioOutline.
    getExamples()) {
        AstNode examplesNode = new AstNode(examples,
        childNode);
        TableRow headerRow = examples.getTableHeader();
        AstNode headerNode = new AstNode(headerRow,
        examplesNode);
        nodeMap.put(headerRow.getLocation().getLine(),
        headerNode);
        for (int i = 0; i < examples.getTableBody().
        size(); ++i) {
            TableRow examplesRow = examples.
            getTableBody().get(i);
            Node rowNode = new ExamplesRowWrapperNode(examp
            lesRow, i);
            AstNode expandedScenarioNode = new
            AstNode(rowNode, examplesNode);
            nodeMap.put(examplesRow.getLocation().
            getLine(), expandedScenarioNode);
        }
    }
}
```

```
    static class ExamplesRowWrapperNode extends Node {
        final int bodyRowIndex;

        ExamplesRowWrapperNode(Node examplesRow, int
        bodyRowIndex) {
            super(examplesRow.getLocation());
            this.bodyRowIndex = bodyRowIndex;
        }
    }

    static class AstNode {
        final Node node;
        final AstNode parent;

        AstNode(Node node, AstNode parent) {
            this.node = node;
            this.parent = parent;
        }
    }
}
```

Listing 11-7. URLOutputStream.java

```
package com.taf.testautomation.cucumber;

import io.cucumber.core.gherkin.vintage.internal.gherkin.deps.
com.google.gson.Gson;

import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Collections;
import java.util.Map;
import java.util.stream.Collectors;

import static java.nio.charset.StandardCharsets.UTF_8;
```

```
class URLOutputStream extends OutputStream {
    private final URL url;
    private final String method;
    private final int expectedResponseCode;
    private final OutputStream out;
    private final HttpURLConnection urlConnection;

    URLOutputStream(URL url) throws IOException {
        this(url, "PUT", Collections.emptyMap(), 200);
    }

    private URLOutputStream(URL url, String method, Map<String,
    String> headers, int expectedResponseCode) throws
    IOException {
        this.url = url;
        this.method = method;
        this.expectedResponseCode = expectedResponseCode;
        if (url.getProtocol().equals("file")) {
            File file = new File(url.getFile());
            ensureParentDirExists(file);
            out = new FileOutputStream(file);
            urlConnection = null;
        } else if (url.getProtocol().startsWith("http")) {
            urlConnection = (HttpURLConnection) url.
            openConnection();
            urlConnection.setRequestMethod(method);
            urlConnection.setDoOutput(true);
            for (Map.Entry<String, String> header : headers.
            entrySet()) {
                urlConnection.setRequestProperty(header.
                getKey(), header.getValue());
            }
```

```
            out = urlConnection.getOutputStream();
        } else {
            throw new IllegalArgumentException("URL Scheme
            must be one of file,http,https. " + url.
            toExternalForm());
        }
    }

    private void ensureParentDirExists(File file) throws
    IOException {
        if (file.getParentFile() != null && !file.
        getParentFile().isDirectory()) {
            boolean ok = file.getParentFile().mkdirs() || file.
            getParentFile().isDirectory();
            if (!ok) {
                throw new IOException("Failed to create
                directory " + file.getParentFile().
                getAbsolutePath());
            }
        }
    }

    @Override
    public void write(byte[] buffer, int offset, int count)
    throws IOException {
        out.write(buffer, offset, count);
    }

    @Override
    public void write(byte[] buffer) throws IOException {
        out.write(buffer);
    }
```

```
@Override
public void write(int b) throws IOException {
    out.write(b);
}

@Override
public void flush() throws IOException {
    out.flush();
}

@Override
public void close() throws IOException {
    try {
        if (urlConnection == null) {
            return;
        }

        int responseCode = urlConnection.getResponseCode();
        if (responseCode == expectedResponseCode) {
            return;
        }

        try {
            urlConnection.getInputStream().close();
            throw new IOException(String.format("
            Expected response code: %d. Got: %d",
            expectedResponseCode, responseCode));
        } catch (IOException expected) {
            InputStream errorStream = urlConnection.
            getErrorStream();
            if (errorStream != null) {
                throw createResponseException(responseCode,
                expected, errorStream);
```

```
                } else {
                    throw expected;
                }
            }
        } finally {
            out.close();
        }
    }

    private ResponseException createResponseException(int
    responseCode, IOException expected, InputStream
    errorStream) throws IOException {
        try (BufferedReader br = new BufferedReader(new
        InputStreamReader(errorStream, UTF_8))) {
            String responseBody = br.lines().collect(Collectors
            .joining(System.lineSeparator()));
            String contentType = urlConnection.getHeaderField("
            Content-Type");
            if (contentType == null) {
                contentType = "text/plain";
            }
            return new ResponseException(responseBody,
            expected, responseCode, contentType);
        }
    }

    class ResponseException extends IOException {
        private static final long serialVersionUID =
        7564052957642239835L;
        private final Gson gson = new Gson();
        private final int responseCode;
        private final String contentType;
```

```
        public ResponseException(String responseBody,
        IOException cause, int responseCode, String contentType) {
            super(responseBody, cause);
            this.responseCode = responseCode;
            this.contentType = contentType;
        }

        @Override
        public String getMessage() {
            if (contentType.equals("application/json")) {
                @SuppressWarnings("rawtypes")
                Map map = gson.fromJson(super.getMessage(),
                Map.class);
                if (map.containsKey("error")) {
                    return getMessage0(map.get("error").
                    toString());
                } else {
                    return getMessage0(super.getMessage());
                }
            } else {
                return getMessage0(super.getMessage());
            }
        }

        private String getMessage0(String message) {
            return String.format("%s %s\nHTTP %d\n%s", method,
            url, responseCode, message);
        }
    }
}
```

In all the preceding files, I have replaced a few deprecated packages. In addition, in the `ExtentCucumberAdapter updateResult(Result result)` method, I instantiate the `ExtentHtmlReporter` class using `REPORT_PATH` and load `REPORT_CONFIG` using the `loadXMLConfig` method.

Improving Extent Report to Print Data Tables

A standard Extent report does not print data tables. To make the report print our data table I have updated the `createTestStep(PickleStepTestStep testStep)` method by adding at the end the snippet shown in Listing 11-8.

Listing 11-8. Code Snippet to Print Data Table in Extent Report

```
if (testStep.getStep().getText().contains("following")) {
    stepTestThreadLocal.get().pass(MarkupHelper.createTable
    (dataTable).getMarkup());
}
```

So now whenever we use a data table in the feature file, we need to have the corresponding step or the following step use the keyword `following`. At that step, our data table will be printed in the report. Now, let's see how to put an Extent report in different folders for each test suite.

Creating Separate Extent Report for Each Test Suite

Suppose we have multiple test suites, and we want to generate an Extent report for each test suite in separate subfolders within the `htmlreport` folder. This is useful, as otherwise the output of the second run would override that of the first run.

Let's add the method `fileCopy(String inputFile, String outputFile)` to the `FileUtil` class we created earlier (Listing 9-6) so that the class now looks like Listing 11-9.

Listing 11-9. Updated FileUtil.java

```
package com.taf.testautomation.utilities.fileutil;

import lombok.extern.slf4j.Slf4j;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import java.io.*;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.stream.Collectors;
import java.util.stream.Stream;

import static com.taf.testautomation.utilities.excelutil.ExcelU
til.getCustomProperties;

@Slf4j
public class FileUtil {

    private LinkedHashMap<String, String> customSettings = new
    LinkedHashMap<>();
    private static final String INPUT_XML_BASE_PATH = "src/
    test/resources/testdata/xml";
```

```
    private static final String INPUT_TXT_BASE_PATH = "src/
    test/resources/testdata/txt";
    private static final String XML_FILENAME = "data.xml";
    private static final String DATA_FILENAME = "data.txt";
    private static FileOutputStream fos = null;

    public FileUtil() {
    }

    public Map<String, String> getCustomSettings() {
        String filePath = getInputFilePath();
        try {
            parseXML(filePath, customSettings);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return customSettings;
    }

    public String getCustomValues(String key) {
        String filePath = getInputFilePath();
        String value = "";
        try {
            value = parseFileLines(filePath, key);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return value;
    }

    private String getInputFilePath() {
        if (getCustomProperties().get("isAndroid").
       equals("true")) {
```

```
            return INPUT_XML_BASE_PATH + XML_FILENAME;
        } else {
            return INPUT_TXT_BASE_PATH + DATA_FILENAME;
        }
    }

    private void parseXML(String filePath,
    LinkedHashMap<String, String> customProperties) throws
    Exception {
        File fXmlFile = new File(filePath);
        DocumentBuilderFactory dbFactory = DocumentBuilderFacto
        ry.newInstance();
        DocumentBuilder dBuilder = dbFactory.
        newDocumentBuilder();
        Document doc = dBuilder.parse(fXmlFile);
        doc.getDocumentElement().normalize();
        log.info("Root element :" + doc.getDocumentElement().
        getNodeName());
        NodeList nList = doc.getElementsByTagName("string");
        for (int i = 0; i < nList.getLength(); i++) {
            Node nNode = nList.item(i);
            Element eElement = (Element) nNode;
            String key = eElement.getAttribute("name");
            String value = nNode.getTextContent();
            customProperties.put(key, value);
        }
    }

    private String parseFileLines(String filePath,
    String key) {
        String strLine = "";
        try {
```

```
            Stream<String> lines = Files.lines(Paths.get(f
            ilePath));
            strLine = lines.filter(s -> s.contains(key)).
            collect(Collectors.toList()).get(0);
        } catch (Exception e) {
            e.printStackTrace();
        }
        if (getCustomProperties().get("isAndroid").
        equals("true")) {
            return strLine.substring(strLine.indexOf(">") + 1,
            strLine.lastIndexOf("<"));
        } else {
            return strLine.substring(strLine.indexOf("=") + 3,
            strLine.indexOf(";") - 1);
        }
    }

    public static void fileCopy(String inputFile, String
    outputFile) {
        String cleanLine = "";

        try {
            fos = new FileOutputStream(outputFile);
            FileInputStream fileInputStream = new
            FileInputStream(inputFile);
            BufferedReader br = new BufferedReader(new InputStr
            eamReader(fileInputStream));
            String strLine = null;
            int linesCounter = 0;
            while ((strLine = br.readLine()) != null) {// read
            every line in the file
                cleanLine += strLine + "\n";
            }
```

```
            char[] stringToCharArray = cleanLine.toCharArray();
            for (char ch : stringToCharArray)
                fos.write(ch);
            fos.close();
        } catch (IOException e) {
        }
    }
}
```

Now, in the AboutAppTest runner file, add an @AfterClass method, as shown in Listing 11-10. The method has to be annotated @AfterClass as the Extent report does not get generated until the runner file is executed.

Listing 11-10. Updated AboutAppTest Runner File

```
package com.taf.testautomation.cucumber.aboutapp;

import com.taf.testautomation.utilities.fileutil.FileUtil;
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.AfterClass;
import org.junit.runner.RunWith;

import java.io.File;

import static com.taf.testautomation.utilities.excelutil.Excel
Util.getCustomProperties;

@RunWith(Cucumber.class)
@CucumberOptions(features = "src/test/resources/features", glue
= {"com.taf.testautomation.cucumber.aboutapp"}, monochrome =
true, plugin = {"html:build/cucumber-html-report-normal",
        "json:build/cucumber.json", "com.taf.testautomation.
        cucumber.ExtentCucumberAdapter:"}, tags =
        {"@AboutApp"})
```

```
public class AboutAppTest {
    @AfterClass
    public static void sendExtentReport() {
        String folder = new Object() {
        }.getClass().getName();
        folder = folder.substring(folder.lastIndexOf('.') + 1,
        folder.indexOf('$'));
        String htmlFilePath = getCustomProperties().
        get("reportPrefix") + "test-result/htmlreport/"
        + folder;
        String htmlFile = htmlFilePath + "/ExtentHtml.html";
        String folderExists = new File(htmlFilePath).mkdir() ?
        "Folder Created" : "Folder Exists";
        FileUtil.fileCopy("test-result/htmlreport/ExtentHtml.
        html", new File(htmlFile).getPath());
    }
}
```

Now, after `ExtentHtml.html` is generated, it is also copied into a subfolder named after this test.

Summary

In this chapter, we learned how to implement Allure reports for both BDD and non-BDD implementations. These are standard HTML reports that are widely used. In the next chapter, we will learn how to add the capability to create a customized PDF report with screenshots.

CHAPTER 12

Creating a Pdf Report with Screenshots

In the previous chapter, we learned how to create Allure and Extent HTML reports for our test suite. While the Allure report is for both BDD and TDD, the Extent report is for Cucumber. In this chapter, we will learn how to create a PDF report that can be customized to your organization. First, let's write a utility class for this purpose.

Creating a PdfUtil Class to Generate Report for Each Test Suite

In Chapter 6, we created a `PdfUtil` class within the `src/main/java/com/taf/testautomation/utilities/pdfutil` folder. Now we will write a few methods using the `itextpdf` plugin, as shown in Listing 12-1.

Listing 12-1. PdfUtil.java

```
package com.taf.testautomation.utilities.pdfutil;

import com.itextpdf.text.*;
import com.itextpdf.text.pdf.*;

import java.io.*;
```


K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_12

```
import java.net.URI;
import java.net.URL;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.IntStream;

import static com.taf.testautomation.utilities.excelutil.ExcelU
til.getCustomProperties;

public class PdfUtil extends PdfPageEventHelper {

    public void getPdfFromImageList(List<String> props,
    List<String> image, File pdfFile) {
        try {
            int count = 1;
            int lineCount = 30;
            Document document = new Document();
            PdfWriter writer = PdfWriter.getInstance(document,
            new FileOutputStream(pdfFile));
            document.open();
            addHeaderFooter(document, writer, count);
            document.addAuthor("Koushik Das");
            document.addCreationDate();
            document.addCreator("Koushik Das");
            document.addTitle("Test ScreenShots");
            document.addSubject("Test ScreenShots");
            Paragraph pg = new Paragraph();
            pg.add("\n");
            pg.add("\n Test Name:      " + props.get(0));
            pg.add("\n ========================================
            ====================");
```

```
pg.add("\n Time Zone:      " + props.get(1));
pg.add("\n Report Date:      " + props.get(2));
pg.add("\n Report Time:      " + props.get(3));
pg.add("\n Device Name:      " + props.get(4));
pg.add("\n Device OS:      " + props.get(5));
pg.add("\n Phone Model:      " + props.get(6));
pg.add("\n Device Mfg:      " + props.get(7));
pg.add("\n OS Version:      " + props.get(8));
pg.add("\n Device SN:      " + props.get(9));
pg.add("\n Build Number:      " + props.get(10));
IntStream.range(0, 33).forEach(i -> pg.add("\n"));
pg.add("\n ======================================
====================");
document.add(pg);
for (String str : image) {
    document.add(new Paragraph(str.substring(str.
    lastIndexOf("/") + 1, str.indexOf("."))));
    Image image1 = Image.getInstance(str);
    image1.scaleAbsolute(200, 200);
    document.add(image1);
    Paragraph para = new Paragraph();
    count++;
    if ((count % 2) == 0) lineCount = 33;
    else lineCount = 28;
    IntStream.range(0, lineCount).forEach(i ->
    para.add("\n"));
    para.add("\n ================================
    ==========================");
    addHeaderFooter(document, writer, count);
    document.add(para);
}
```

```
            document.close();
            writer.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void addHeaderFooter(Document doc, PdfWriter
    writer, int count) {
        doc.setPageCount(count);
        onStartPage(doc, writer);
        onEndPage(doc, writer);
    }

    private void onStartPage(Document doc, PdfWriter writer) {
        int numberOfPages = doc.getPageNumber();
        ColumnText.showTextAligned(writer.getDirectContent(),
        Element.ALIGN_CENTER, new Phrase("page " +
        numberOfPages), 550, 800, 0);
    }

    private void onEndPage(Document doc, PdfWriter writer) {
        ColumnText.showTextAligned(writer.getDirectContent(),
        Element.ALIGN_CENTER, new Phrase("www.xxx.com"),
        550, 30, 0);
    }

    public static void getPdfFromImage(String image, String
    pdfFile) {
        try {
            Document document = new Document();
            PdfWriter writer = PdfWriter.getInstance(document,
            new FileOutputStream(pdfFile));
```

```
            document.open();
            document.addAuthor("Koushik Das");
            document.addCreationDate();
            document.addCreator("Koushik Das");
            document.addTitle("Test ScreenShots");
            document.addSubject("Test ScreenShots");
            document.add(new Paragraph("Screenshots"));
            Image image1 = Image.getInstance(image);
            image1.scaleAbsolute(200, 200);
            document.add(image1);
            document.close();
            writer.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void mergePdf(File pdfFile) {
        try {
            String folder = getCustomProperties().
            get("reportPrefix") + "test-result/pdfreport";
            List<String> inputPdfList = new ArrayList<>();
            Files.newDirectoryStream(Paths.get(folder),
                    path -> path.toString().endsWith(".
                    pdf")).forEach(filePath -> inputPdfList.
                    add(filePath.toString()));

            Document document = new Document();
            PdfCopy copy = new PdfCopy(document, new
            FileOutputStream(pdfFile));
            document.open();
```

```
            for (String file : inputPdfList) {
                if(!file.equals(getCustomProperties().
                get("mergedReport"))) {
                    File pdf = new File(file);
                    URI uri = pdf.toURI();
                    URL url = uri.toURL();
                    PdfReader reader = new PdfReader(url);
                    copy.addDocument(reader);
                    copy.freeReader(reader);
                    reader.close();
                }
            }
            document.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void getPdfFromHtml(String htmlSource, String
    pdfFile) {
        try {

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

The method `getPdfFromImageList(List<String> props, List<String> image, File pdfFile)` generates a PDF file with a list of device properties and images and saves this data in `pdfFile`. It also adds a page number header and a footer that reads `www.xxxx.com`, where `x` could be an organization's website. We will write methods to read the device

properties for Android and iOS in Chapter 17. You can print the device properties in an iterative loop using `IntStream.range(0, props.size())`, but you need to update the number of subsequent `newlines` accordingly to maintain the format. You can also parameterize the author name, title, and so forth in a list of strings. I have kept the number of arguments to three as part of our best practices, mentioned in Chapter 1.

We used the method `getPdfFromImage(String image, String pdfFile)` in the `MobileBaseActionScreen` class to capture a screenshot and put it in a PDF file. Now, let's see how to pass parameters to the preceding method from our test suite.

Passing Parameters to PdfUtil from a Test Suite

Let's create a class named `AppiumUtil` within the `src/main/java/com/taf/testautomation` folder. Define the empty method `public String getDeviceProperties(String devProp) { return "string" }` in the class. We will implement this method in Chapter 17.

Now, go to the test suite `AboutAppTestSuite` (Listing 8-2) and add a new method at the end to plug in the report module. I have created an empty list type `imageList` in the test suite. We will populate this `imageList` in the next chapter. So, our test suite should now look like Listing 12-2.

Listing 12-2. AboutAppTestSuite.java

```
package com.taf.testautomation.uitests;

import com.taf.testautomation.BaseTest;
import com.taf.testautomation.annotations.*;
import com.taf.testautomation.screens.AboutAppScreen;
import com.taf.testautomation.utilities.excelutil.ExcelUtil;
import com.taf.testautomation.utilities.jsonutil.JsonUtil;
```

```
import com.taf.testautomation.utilities.pdfutil.PdfUtil;
import com.taf.testautomation.workflows.ScreenNavigation;
import com.taf.testautomation.devicemanagement.AppiumUtil;
import io.qameta.allure.*;
import org.assertj.core.api.SoftAssertions;
import org.junit.jupiter.api.*;

import java.io.File;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
@Epic("xxxx")
@Feature("About App Page Layout")
public class AboutAppTestSuite extends BaseTest {

    private AboutAppScreen aboutAppScreen;
    private JsonUtil jsonUtil;
    private ExcelUtil excelUtil = new ExcelUtil();
    private AppiumUtil appiumUtil;
    protected String testStatus = "";
    private static final String SCREEN_NAME = "aboutAppScreen";
    private static final String TEST_NAME = "AboutApp-Screen-
    Verification-";
    private static int i = 0, j = 0;
    private static List<String> imageList = new ArrayList<>();

    @BeforeAll
    @Override
    public void setUp() throws Exception {
```

```
    super.setUp();
    if (getCustomProperties().get("loadExcel").
    equals("Y")) {
        excelUtil.generateJsonFilesFromExcel1();
    }
    jsonUtil = new JsonUtil();
    appiumUtil = new AppiumUtil(getSession().
    getAppiumDriver());
}

@BeforeEach
public void navigateToScreen() {
    log("Navigating to " + SCREEN_NAME);
    if (testStatus.isEmpty()) {
        aboutAppScreen = new ScreenNavigation(session).get
        AboutAppScreenFromAccountCreationScreen(getCustomP
        roperties().get("username"), getCustomProperties().
        get("password"));
    } else {
        log("User in About App screen already");
    }

}

@AfterAll
@Override
public void tearDown() throws Exception {
    super.tearDown();
}

@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
```

```
    @Description("xxxx: Verify that the AboutAppScreen Title is
    displayed")
    @Test
    @Order(1)
    @Smoke
    @Regression
    @SIT
    @AT
    public void testScenario1() {
        String tcName = new Object() {
        }.getClass().getEnclosingMethod().getName();
        log("Test Name" + tcName);

        SoftAssertions.assertSoftly(
                softAssertions -> {
                    softAssertions.assertThat(aboutAppScreen.
                    isScreenTitleDisplayed()).as("The Screen
                    title is displayed").isTrue();
                }
        );
        testStatus = aboutAppScreen.isScreenTitleDisplayed() ?
        "Passed" : "Failed";
        updateTCPassCount();
    }

    @Severity(SeverityLevel.CRITICAL)
    @Issue("xxxx")
    @DisplayName("xxxx")
    @Description("xxxx: Verify that the App Logo is displayed")
    @Test
    @Order(2)
    @Smoke
    public void testScenario2() {
```

```
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    SoftAssertions.assertSoftly(
            softAssertions -> {
                softAssertions.assertThat(aboutAppScreen.
                isAppLogoDisplayed()).as("The App Logo is
                displayed").isTrue();
            }
    );
    testStatus = aboutAppScreen.isAppLogoDisplayed() ?
    "Passed" : "Failed";
    updateTCPassCount();
}

@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the App Name is displayed")
@Test
@Order(3)
@Regression
public void testScenario3() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    SoftAssertions.assertSoftly(
            softAssertions -> {
                softAssertions.assertThat(aboutAppScreen.
                isAppNameDisplayed()).as("The App Name is
                displayed").isTrue();
            }
```

```
        );
        testStatus = aboutAppScreen.isAppNameDisplayed() ?
        "Passed" : "Failed";
        updateTCPassCount();
    }

    @Severity(SeverityLevel.CRITICAL)
    @Issue("xxxx")
    @DisplayName("xxxx")
    @Description("xxxx: Verify that the App Version is
    displayed")
    @Test
    @Order(4)
    @SIT
    public void testScenario4() {
        String tcName = new Object() {
        }.getClass().getEnclosingMethod().getName();
        log("Test Name" + tcName);

        SoftAssertions.assertSoftly(
                softAssertions -> {
                    softAssertions.assertThat(aboutAppScreen.
                    isAppVersionDisplayed("xxxx")).as("The App
                    Version is displayed").isTrue();
                }
        );
        testStatus = aboutAppScreen.isAppVersionDisplayed
        ("xxxx") ? "Passed" : "Failed";
        updateTCPassCount();
    }

    @Severity(SeverityLevel.CRITICAL)
    @Issue("xxxx")
```

```
@DisplayName("xxxx")
@Description("xxxx: Verify that the App images are
displayed")
@Test
@Order(5)
@AT
public void testScenario5() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    SoftAssertions.assertSoftly(
            softAssertions -> {
                softAssertions.assertThat(aboutAppScreen.
                areAppImagesDisplayed()).as("The App images
                are displayed").isTrue();
            }
    );
    testStatus = aboutAppScreen.areAppImagesDisplayed() ?
    "Passed" : "Failed";
    updateTCPassCount();
}

@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Create Report")
@Test
@Order(6)
@Smoke
@Regression
@SIT
```

```
    @AT
    public void create_pdf_report() {
        String tcName = new Object() {
        }.getClass().getEnclosingMethod().getName();
        log("Test Name" + tcName);

        appiumUtil = new AppiumUtil(getSession().
        getAppiumDriver());
        String pdfFile = getCustomProperties().
        get("reportPrefix") + "test-result/pdfreport/" +
        TEST_NAME + "Report.pdf";
        String testCase = TEST_NAME + "Screenshots:";
        String timeZone = appiumUtil.getDeviceProperties
        ("TimeZone");
        String reportDate = appiumUtil.getDeviceProperties
        ("ReportDate");
        String reportTime = appiumUtil.getDeviceProperties
        ("ReportTime");
        String deviceName = appiumUtil.getDeviceProperties
        ("Name");
        String deviceOS = appiumUtil.getDeviceProperties("OS");
        String deviceModel = appiumUtil.getDeviceProperties
        ("Model");
        String deviceMfg = appiumUtil.getDeviceProperties
        ("Manufacturer");
        String osVersion = appiumUtil.getDeviceProperties
        ("Version");
        String deviceSN = appiumUtil.getDeviceProperties
        ("Serial_Number");
        String buildNumber = getCustomProperties().get("build");
        Stream<String> propStream = Stream.of(testCase,
        timeZone, reportDate, reportTime, deviceName,
```

```
        deviceOS, deviceModel, deviceMfg, osVersion, deviceSN,
        buildNumber);
        List<String> propList = propStream.collect(Collectors.
        toList());

        String update = j + " of " + i + " Passed";
        new PdfUtil().getPdfFromImageList(propList, imageList,
        new File(pdfFile));
        PdfUtil.mergePdf(new File(getCustomProperties().
        get("mergedReport")));
    }

    private void updateTCPassCount() {
        i++;
        if (testStatus.equals("Passed")) j++;
    }

}
```

Finally, don't forget to update the BDD step definition file. Our last step definition is as shown in Listing 12-3.

Listing 12-3. Cucumber Step to Create Report

```
@Then("close application and update HP QC test run")
public void close_application_and_update_HP_QC_test_run()
throws Exception {
    create_pdf_report();
    tearDown();
}
```

The corresponding PDF (with empty list of images) should be as in Figure 12-1. I have redacted some of my device-specific information with "XXXX."

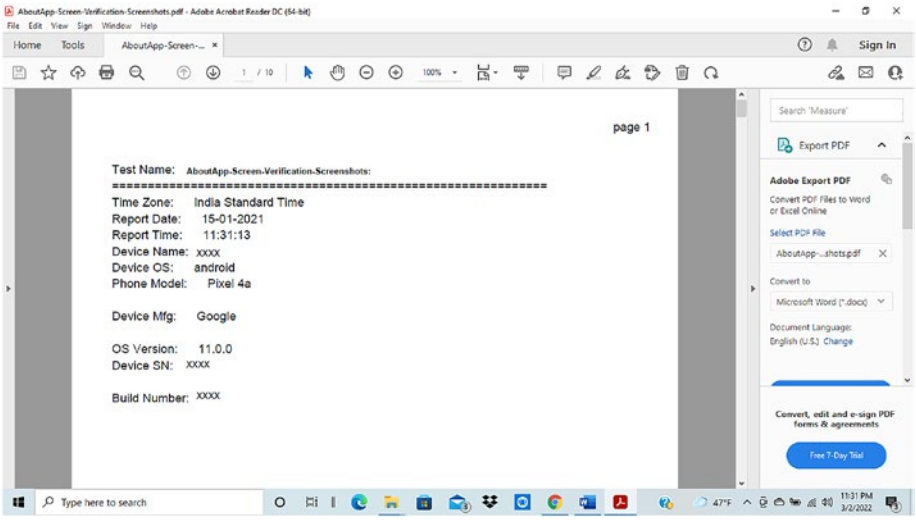


Figure 12-1. *Sample PDF report*

Now let's discuss the method used to merge multiple PDF files.

Merging Multiple PDFs

The method `mergePdf(File pdfFile)` in the `PdfUtil` class merges multiple PDF files within the `test-result/pdfreport` folder into one file, `pdfFile`. It ignores the merged file as input if already present. We then need to call this method only if we must merge multiple PDF test reports generated by running multiple test suites.

Summary

In this chapter, we created a PDF report that could be generated for both BDD and TDD executions. The report UI is customizable, and I encourage the reader to tweak the UI as needed. In the next chapter, we will discuss a few ways to generate and save screenshots that could be used in the reports.

CHAPTER 13

Enhancing the Framework: Screenshots

In the previous chapter, we created a PDF report for a test execution. We created an empty list type `imageList` in the test suite to store the images. In this chapter, we will discuss a few methods that can be used to generate and save screenshots. We will start by looking at the method we created in the `MobileBaseActionScreen` class.

Creating Screenshot and Saving in Default Location

We have already created methods in the `MobileBaseActionScreen` class that will take screenshots. The method that saved the screenshot in the project root can be used by any page object method to take a screenshot and compare it with a reference image. We will discuss this in the "Image Comparison Util" section of Appendix A. The code for this method is reproduced in Listing 13-1.

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_13

Listing 13-1. takeScreenShot Method

```
/**
 * Takes a screenshot of the current screen
 * and writes it into the project path
 */
public void takeScreenShot() throws IOException {
    File scrFile = ((TakesScreenshot) getSession().
    getAppiumDriver()).getScreenshotAs(OutputType.FILE);
    BufferedImage image = ImageIO.read(scrFile);
    File outputFile = new File("screenshot.png");
    ImageIO.write(image, "png", outputFile);
}
```

Now, let's see how to create screenshots and save them in various locations.

Creating Screenshots and Saving in Various Locations

For this purpose, we will use the method shown in Listing 13-2 from the `MobileBaseActionScreen` class (Listing 6-4).

Listing 13-2. takeScreenShot Overloaded Method

```
/**
 * Takes a screenshot of the current screen
 * and writes it into the project path with a given file name
 */
public void takeScreenShot(String fileName) throws
IOException {
```

```
    File scrFile = ((TakesScreenshot) getSession().
    getAppiumDriver()).getScreenshotAs(OutputType.FILE);
    BufferedImage image = ImageIO.read(scrFile);
    File outputFile = new File(fileName);
    ImageIO.write(image, "png", outputFile);
}
```

We will use the following code in our test suites to gather screenshots from various verification points and store them in the `test-result/screenshots` folder. We will also use these screenshots to populate the `imageList` that we created in our test suite `AboutAppTestSuite`. It is now time to update our test suite to add this capability, as shown in Listing 13-3.

Listing 13-3. Updated AboutAppTestSuite.java

```
package com.taf.testautomation.uitests;

import com.taf.testautomation.BaseTest;
import com.taf.testautomation.annotations.*;
import com.taf.testautomation.screens.AboutAppScreen;
import com.taf.testautomation.utilities.excelutil.ExcelUtil;
import com.taf.testautomation.utilities.jsonutil.JsonUtil;
import com.taf.testautomation.utilities.pdfutil.PdfUtil;
import com.taf.testautomation.workflows.ScreenNavigation;
import com.taf.testautomation.devicemanagement.AppiumUtil;
import io.qameta.allure.*;
import org.assertj.core.api.SoftAssertions;
import org.junit.jupiter.api.*;

import java.io.File;
import java.util.ArrayList;
import java.util.List;
```

```
import java.util.stream.Collectors;
import java.util.stream.Stream;

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
@Epic("xxxx")
@Feature("About App Page Layout")
public class AboutAppTestSuite extends BaseTest {

    private AboutAppScreen aboutAppScreen;
    private JsonUtil jsonUtil;
    private ExcelUtil excelUtil = new ExcelUtil();
    private AppiumUtil appiumUtil;
    protected String testStatus = "";
    private static final String SCREEN_NAME = "aboutAppScreen";
    private static final String TEST_NAME = "AboutApp-Screen-
    Verification-";
    private static int i = 0, j = 0;
    private static List<String> imageList = new ArrayList<>();

    @BeforeAll
    @Override
    public void setUp() throws Exception {
        super.setUp();
        if (getCustomProperties().get("loadExcel").equals("Y")) {
            excelUtil.generateJsonFilesFromExcel1();
        }
        jsonUtil = new JsonUtil();
        appiumUtil = new AppiumUtil(getSession().getAppium
        Driver());
    }
```

```
@BeforeEach
public void navigateToScreen() {
    log("Navigating to " + SCREEN_NAME);
    if (testStatus.isEmpty()) {
        aboutAppScreen = new ScreenNavigation(session).get
        AboutAppScreenFromAccountCreationScreen(getCustomP
        roperties().get("username"), getCustomProperties().
        get("password"));
    } else {
        log("User in About App screen already");
    }
}

@AfterAll
@Override
public void tearDown() throws Exception {
    super.tearDown();
}

@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the AboutAppScreen Title is
displayed")
@Test
@Order(1)
@Smoke
@Regression
@SIT
@AT
public void testScenario1() {
```

```
        String tcName = new Object() {
        }.getClass().getEnclosingMethod().getName();
        log("Test Name" + tcName);

        SoftAssertions.assertSoftly(
                softAssertions -> {
                    softAssertions.assertThat(aboutAppScreen.
                    isScreenTitleDisplayed()).as("The Screen
                    title is displayed").isTrue();
                }
        );
        testStatus = aboutAppScreen.isScreenTitleDisplayed() ?
        "Passed" : "Failed";
        updateTCPassCount();
        try {
            aboutAppScreen.takeScreenShot("test-result/
            screenshots/" + tcName + "--" + testStatus + ".png");
        } catch (Exception e) {
            e.printStackTrace();
        }
        imageList.add("test-result/screenshots/" + tcName +
        "--" + testStatus + ".png");
    }

    @Severity(SeverityLevel.CRITICAL)
    @Issue("xxxx")
    @DisplayName("xxxx")
    @Description("xxxx: Verify that the App Logo is displayed")
    @Test
    @Order(2)
    @Smoke
```

```
public void testScenario2() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    SoftAssertions.assertSoftly(
            softAssertions -> {
                softAssertions.assertThat(aboutAppScreen.
                isAppLogoDisplayed()).as("The App Logo is
                displayed").isTrue();
            }
    );
    testStatus = aboutAppScreen.isAppLogoDisplayed() ?
    "Passed" : "Failed";
    updateTCPassCount();
    try {
        aboutAppScreen.takeScreenShot("test-result/
        screenshots/" + tcName + "--" + testStatus + ".png");
    } catch (Exception e) {
        e.printStackTrace();
    }
    imageList.add("test-result/screenshots/" + tcName +
    "--" + testStatus + ".png");
}

@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the App Name is displayed")
@Test
@Order(3)
@Regression
```

```
    public void testScenario3() {
        String tcName = new Object() {
        }.getClass().getEnclosingMethod().getName();
        log("Test Name" + tcName);

        SoftAssertions.assertSoftly(
                softAssertions -> {
                    softAssertions.assertThat(aboutAppScreen.
                    isAppNameDisplayed()).as("The App Name is
                    displayed").isTrue();
                }
        );
        testStatus = aboutAppScreen.isAppNameDisplayed() ?
        "Passed" : "Failed";
        updateTCPassCount();
        try {
            aboutAppScreen.takeScreenShot("test-result/
            screenshots/" + tcName + "--" + testStatus + ".png");
        } catch (Exception e) {
            e.printStackTrace();
        }
        imageList.add("test-result/screenshots/" + tcName +
        "--" + testStatus + ".png");
    }

    @Severity(SeverityLevel.CRITICAL)
    @Issue("xxxx")
    @DisplayName("xxxx")
    @Description("xxxx: Verify that the App Version is displayed")
    @Test
    @Order(4)
    @SIT
```

```
public void testScenario4() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    SoftAssertions.assertSoftly(
            softAssertions -> {
                softAssertions.assertThat(aboutAppScreen.
                isAppVersionDisplayed("xxxx")).as("The App
                Version is displayed").isTrue();
            }
    );
    testStatus = aboutAppScreen.isAppVersionDisplayed
    ("xxxx") ? "Passed" : "Failed";
    updateTCPassCount();
    try {
        aboutAppScreen.takeScreenShot("test-result/
        screenshots/" + tcName + "--" + testStatus + ".png");
    } catch (Exception e) {
        e.printStackTrace();
    }
    imageList.add("test-result/screenshots/" + tcName +
    "--" + testStatus + ".png");
}

@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the App images are displayed")
@Test
@Order(5)
@AT
```

```
    public void testScenario5() {
        String tcName = new Object() {
        }.getClass().getEnclosingMethod().getName();
        log("Test Name" + tcName);

        SoftAssertions.assertSoftly(
                softAssertions -> {
                    softAssertions.assertThat(aboutAppScreen.
                    areAppImagesDisplayed()).as("The App images
                    are displayed").isTrue();
                }
        );
        testStatus = aboutAppScreen.areAppImagesDisplayed() ?
        "Passed" : "Failed";
        updateTCPassCount();
        try {
            aboutAppScreen.takeScreenShot("test-result/
            screenshots/" + tcName + "--" + testStatus + ".png");
        } catch (Exception e) {
            e.printStackTrace();
        }
        imageList.add("test-result/screenshots/" + tcName +
        "--" + testStatus + ".png");
    }

    @Severity(SeverityLevel.CRITICAL)
    @Issue("xxxx")
    @DisplayName("xxxx")
    @Description("xxxx: Create Report")
    @Test
    @Order(6)
    @Smoke
```

```
@Regression
@SIT
@AT
public void create_pdf_report() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    appiumUtil = new AppiumUtil(getSession().getAppium
    Driver());
    String pdfFile = getCustomProperties().get
    ("reportPrefix") + "test-result/pdfreport/" + TEST_NAME
    + "Report.pdf";
    String testCase = TEST_NAME + "Screenshots:";
    String timeZone = appiumUtil.getDeviceProperties
    ("TimeZone");
    String reportDate = appiumUtil.getDeviceProperties
    ("ReportDate");
    String reportTime = appiumUtil.getDeviceProperties
    ("ReportTime");
    String deviceName = appiumUtil.
    getDeviceProperties("Name");
    String deviceOS = appiumUtil.getDeviceProperties("OS");
    String deviceModel = appiumUtil.getDeviceProperties
    ("Model");
    String deviceMfg = appiumUtil.getDeviceProperties
    ("Manufacturer");
    String osVersion = appiumUtil.getDeviceProperties("
    Version");
    String deviceSN = appiumUtil.getDeviceProperties
    ("Serial_Number");
```

```
        String buildNumber = getCustomProperties().get("build");
        Stream<String> propStream = Stream.of(testCase,
        timeZone, reportDate, reportTime, deviceName,
        deviceOS, deviceModel, deviceMfg, osVersion, deviceSN,
        buildNumber);
        List<String> propList = propStream.collect(Collectors.
        toList());

        String update = j + " of " + i + " Passed";
        new PdfUtil().getPdfFromImageList(propList, imageList,
        new File(pdfFile));
        PdfUtil.mergePdf(new File(getCustomProperties().
        get("mergedReport")));
    }

    private void updateTCPassCount() {
        i++;
        if (testStatus.equals("Passed")) j++;
    }

}
```

Our test suite now takes screenshots from each verification point and stores them in the designated folder, giving them unique names that reference the test methods from which the screenshots were taken. Each screenshot name is also appended with the test verification status, namely `Passed` or `Failed`. All of these screenshots will be printed in the PDF report generated after the test suite is run.

You will note that if an assertion fails the preceding code will fail to get the screenshot. We get around this problem by putting the screenshot in a `finally {}` block. This ensures that we get screenshots for failed verifications as well. The updated test methods now look like Listing 13-4.

Listing 13-4. Updated Test Methods in AboutAppTestSuite.java

```
@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the AboutAppScreen Title is
displayed")
@Test
@Order(1)
@Smoke
@Regression
@SIT
@AT
public void testScenario1() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    try {
        SoftAssertions.assertSoftly(
                softAssertions -> {
                    softAssertions.assertThat(aboutApp
                    Screen.isScreenTitleDisplayed()).as
                    ("The Screen title is displayed").isTrue();
                }
        );
    } finally {
        testStatus = aboutAppScreen.isScreenTitleDisplayed() ?
        "Passed" : "Failed";
        updateTCPassCount();
        try {
```

```
            aboutAppScreen.takeScreenShot("test-result/
            screenshots/" + tcName + "--" + testStatus + ".png");
        } catch (Exception e) {
            e.printStackTrace();
        }
        imageList.add("test-result/screenshots/" + tcName +
        "--" + testStatus + ".png");
    }
}

@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the App Logo is displayed")
@Test
@Order(2)
@Smoke
public void testScenario2() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    try {
        SoftAssertions.assertSoftly(
                softAssertions -> {
                    softAssertions.assertThat
                    (aboutAppScreen.isAppLogoDisplayed()).
                    as("The App Logo is displayed").isTrue();
                }
        );
```

```
    } finally {
        testStatus = aboutAppScreen.isAppLogoDisplayed() ?
        "Passed" : "Failed";
        updateTCPassCount();
        try {
            aboutAppScreen.takeScreenShot("test-result/
            screenshots/" + tcName + "--" + testStatus + ".png");
        } catch (Exception e) {
            e.printStackTrace();
        }
        imageList.add("test-result/screenshots/" + tcName +
        "--" + testStatus + ".png");
    }
}

@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the App Name is displayed")
@Test
@Order(3)
@Regression
public void testScenario3() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    try {
        SoftAssertions.assertSoftly(
                softAssertions -> {
```

```
                    softAssertions.assertThat(about
                    AppScreen.isAppNameDisplayed()).as
                    ("The App Name is displayed").isTrue();
                }
        );
    } finally {
        testStatus = aboutAppScreen.isAppNameDisplayed() ?
        "Passed" : "Failed";
        updateTCPassCount();
        try {
            aboutAppScreen.takeScreenShot("test-result/
            screenshots/" + tcName + "--" + testStatus + ".png");
        } catch (Exception e) {
            e.printStackTrace();
        }
        imageList.add("test-result/screenshots/" + tcName +
        "--" + testStatus + ".png");
    }
}

@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the App Version is displayed")
@Test
@Order(4)
@SIT
public void testScenario4() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);
```

```
    try {
        SoftAssertions.assertSoftly(
                softAssertions -> {
                    softAssertions.assertThat
                    (aboutAppScreen.isAppVersionDisplayed
                    ("xxxx")).as("The App Version is
                    displayed").isTrue();
                }
        );
    } finally {
        testStatus = aboutAppScreen.isAppVersionDisplayed
        ("xxxx") ? "Passed" : "Failed";
        updateTCPassCount();
        try {
            aboutAppScreen.takeScreenShot("test-result/
            screenshots/" + tcName + "--" + testStatus + ".png");
        } catch (Exception e) {
            e.printStackTrace();
        }
        imageList.add("test-result/screenshots/" + tcName +
        "--" + testStatus + ".png");
    }
}

@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the App images are displayed")
@Test
@Order(5)
@AT
```

```
public void testScenario5() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    try {
        SoftAssertions.assertSoftly(
                softAssertions -> {
                    softAssertions.assertThat
                    (aboutAppScreen.areAppImages
                    Displayed()).as("The App images are
                    displayed").isTrue();
                }
        );
    } finally {
        testStatus = aboutAppScreen.areAppImagesDisplayed() ?
        "Passed" : "Failed";
        updateTCPassCount();
        try {
            aboutAppScreen.takeScreenShot("test-result/
            screenshots/" + tcName + "--" + testStatus + ".png");
        } catch (Exception e) {
            e.printStackTrace();
        }
        imageList.add("test-result/screenshots/" + tcName +
        "--" + testStatus + ".png");
    }
}
```

Creating Screenshot with Page Object Name

Let's see how to take a screenshot at the page-object level and store it in a designated location. Let's say we want to take screenshots of images from the About App page. While this can be done at the test-suite level in our case, the following approach provides an alternative for when we are compiling multiple `assert` statements in one `assertSoftly` method. In this case, to get an individual screenshot for each assertion, we need to take this approach.

To do this, we will write a new method in the `AboutAppScreen` class and call it from within the verification method, which is `areAppImagesDisplayed()`. The updated page object class is shown in Listing 13-5.

Listing 13-5. Updated AboutAppScreen.java

```
package com.taf.testautomation.screens;

import com.taf.testautomation.Session;
import io.appium.java_client.MobileElement;
import io.appium.java_client.pagefactory.*;
import io.qameta.allure.Step;
import org.openqa.selenium.NoSuchElementException;
import org.openqa.selenium.support.PageFactory;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;
import java.util.stream.Stream;

public class AboutAppScreen extends MobileBaseActionScreen {

    public AboutAppScreen(Session session) {
        super(session);
```

```
        PageFactory.initElements(new AppiumField
        Decorator(session.getAppiumDriver()), this);
    }

    private String XPATH_ANDROID = "", XPATH_IOS = "";

    @AndroidFindBy(xpath = "//android.widget.TextView
    [@text='xxxx']")
    @iOSXCUITFindBy(xpath = "//XCUIElementTypeStaticText
    [@name='xxxx'][@label='xxxx']")
    private MobileElement screenTitle;

    @AndroidFindBy(accessibility = "xxxx")
    @iOSXCUITFindBy(accessibility = "xxxx")
    private MobileElement appVersion;

    @HowToUseLocators(androidAutomation = LocatorGroup
    Strategy.ALL_POSSIBLE,
            iOSXCUITAutomation = LocatorGroupStrategy
            .ALL_POSSIBLE)
    @AndroidFindBy(xpath = "//android.widget.TextView
    [@text='xxxx']")
    @AndroidFindBy(accessibility = "xxxx")
    @iOSXCUITFindBy(xpath = "//XCUIElementTypeStaticText
    [@name='xxxx']")
    @iOSXCUITFindBy(accessibility = "xxxx")
    private MobileElement appName;

    @AndroidFindBy(xpath = "//android.widget.TextView
    [@text='xxxx' or @text='xxxx']")
    @iOSXCUITFindBy(xpath = "//XCUIElementTypeStaticText
    [contains(@name,'xxxx') or contains(@value,'xxxx')]")
    private MobileElement copyrightText;
```

```
@AndroidFindBy(xpath = "//android.widget.ImageView
[@index='xxxx']")
@iOSXCUITFindBy(xpath = "//XCUIElementTypeImage
[contains(@name,'xxxx')]")
private MobileElement appLogo;

@AndroidFindBy(xpath = "//android.widget.ImageView")
@iOSXCUITFindBy(xpath = "//XCUIElementTypeImage")
private List<MobileElement> appImages;

/**
 * Validate if Screen Title is displayed
 *
 * @return True if Screen Title is displayed,
 otherwise returns
 * false.
 */
@Step("Verifying the Screen Title is Displayed")
public boolean isScreenTitleDisplayed() {
    return doesElementExist(screenTitle, SMALL_WAIT);
}

/**
 * Validate if App logo is displayed
 *
 * @return True if App logo is displayed, otherwise returns
 * false.
 */
@Step("Verifying the App logo is Displayed")
public boolean isAppLogoDisplayed() {
    fastScrollDownTouchAction();
```

```
        if (getSession().getCustomProperties().get
        ("isAndroid").equals("true")) {
            return doesElementExist(appLogo, SMALL_WAIT);
        } else {
            return appLogo.isEnabled();
        }
    }

    /**
     * Validate if App Name is displayed
     *
     * @return True if App Name is displayed, otherwise returns
     * false.
     */
    @Step("Verifying the App Name is Displayed")
    public boolean isAppNameDisplayed() {
        fastScrollDownTouchAction();
        return doesElementExist(appName, SMALL_WAIT) &&
        doesElementExist(appName, MIN_WAIT);
    }

    /**
     * Validate if App Version is displayed
     *
     * @return True if App Version is displayed,
     otherwise returns
     * false.
     */
    @Step("Verifying the App Version is Displayed")
    public boolean isAppVersionDisplayed(String text) {
        IntStream.range(0, 2).forEach(i -> scrollUpTouch
        Action());
        waitInSeconds(MIN_WAIT);
```

```
    return doesElementExist(appVersion, SMALL_WAIT) &&
    appVersion.getText().contains(text);
}

/**
 * Validate if Copyright text is displayed
 *
 * @return True if Copyright text is displayed,
 otherwise returns
 * false.
 */
@Step("Verifying the Copyright text is Displayed")
public boolean isCopyrightTextDisplayed(String str) {
    setLocatorText(str);
    if (getSession().getCustomProperties().get
    ("isAndroid").equals("true")) {
        try {
            return doesElementExist (getSession().get
            AppiumDriver().findElementByXPath(XPATH_
            ANDROID), MIN_WAIT);
        } catch (NoSuchElementException e) {
            return false;
        }
    } else {
        try {
            fastScrollDownTouchAction();
            return doesElementExist(getSession().getAppium
            Driver().findElementByXPath(XPATH_IOS),
            MIN_WAIT);
        } catch (NoSuchElementException e) {
            return false;
```

```
            }
        }
    }

    /**
     * Validate if App Description is displayed
     *
     * @return True if App Description is displayed,
     otherwise returns
     * false.
     */
    @Step("Verifying the App Description is Displayed")
    public boolean isAppDescriptionDisplayed(String str1,
    String str2) {
        boolean result = true;
        Stream<String> locatorTextStream = Stream.of
        (str1, str2);
        List<String> locatorTextList = locatorTextStream.collect
        (Collectors.toList());
        for (String str : locatorTextList) {
            setLocatorText(str);
            if (getSession().getCustomProperties().
            get("isAndroid").equals("true")) {
                result = result && doesElementExist
                (getSession().getAppiumDriver().findElement
                ByXPath(XPATH_ANDROID), MIN_WAIT);
            } else {
                result = result && doesElementExist
                (getSession().getAppiumDriver().
                findElementByXPath(XPATH_IOS), MIN_WAIT);
            }
        }
```

```
    return result;
}

/**
 * Validate if app images are displayed
 *
 * @return True if app images are displayed,
 otherwise returns
 * false.
 */
@Step("Verifying the app images are Displayed")
public boolean areAppImagesDisplayed() {
    getScreenShot();
    return doesElementExist(appImages.get(0), MIN_WAIT)
    && doesElementExist(appImages.get(1), MIN_WAIT) &&
    doesElementExist(appImages.get(2), MIN_WAIT);
}

private void setLocatorText(String text) {
    XPATH_ANDROID = "//android.widget.TextView[@text='" +
    text + "']";
    XPATH_IOS = "//XCUIElementTypeStaticText[@name='" +
    text + "']";
}

private void getScreenShot() {
    String name = new Object() {
    }.getClass().getName();
    name = name.substring(name.lastIndexOf('.') + 1,
    name.indexOf('$'));
    try {
        this.takeScreenShot("test-result/screenshots/" +
        name + ".png");
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Summary

In this chapter, we created methods to generate and save screenshots to be used to populate a PDF test report. This makes the PDF report valuable and better than the standard HTML reports discussed in Chapter 11. In the next chapter, we will learn how to test multiple apps and different versions of the same app in same test suite.

CHAPTER 14

Testing Multiple Apps and Versions in Same Test Suite

In the previous chapter, we created methods to generate screenshots to be added to PDF reports. In this chapter, we will discuss ways we can test multiple apps and multiple versions of the same app in the same test suite. We will start by discussing a best practice you should follow.

Best Practice You Should Follow

Running multiple app versions, and especially multiple apps, in same test suite is not a good practice. Ideally, if you want to run a test suite against multiple apps or versions, you should run it separately for each of them. It is quite likely that you will need to change locators for the different apps. Also, from the reporting perspective, it makes more sense to have separate runs for separate versions and apps. However, if there are good use cases, like testing app upgrades, then you will need this approach.

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_14

Testing Multiple Versions of App in Same Test Suite

While creating the `Session` class in Chapter 5, we created the field `prevBuild` and used it with the Android desired capabilities to pick the app. Since we store the `uitests.properties` value in a map and then look at the map to read the values, we can always dynamically alter the map values. If we want to test one scenario—say, testScenario3—in a previous build, we need to change the method, as shown in Listing 14-1.

Listing 14-1. Updated testScenario3 in AboutAppTestSuite.java

```
@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the App Name is displayed")
@Test
@Order(3)
@Regression
public void testScenario3() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    if (session.getAppiumDriver().removeApp
    (getCustomProperties().get("appPackage"))) {
        try {
            session.getAppiumDriver().quit();
            session.setPrevBuild("yes");
            session = startDefaultSession();
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }
    }
    try {
        SoftAssertions.assertSoftly(
                softAssertions -> {
                    softAssertions.assertThat(aboutAppScreen.
                    isAppNameDisplayed()).as("The App Name is
                    displayed").isTrue();
                }
        );
    } finally {
        testStatus = aboutAppScreen.isAppNameDisplayed() ?
        "Passed" : "Failed";
        updateTCPassCount();
        try {
            aboutAppScreen.takeScreenShot("test-result/
            screenshots/" + tcName + "--" + testStatus + ".png");
        } catch (Exception e) {
            e.printStackTrace();
        }
        imageList.add("test-output/ScreenShots/" + tcName +
        "--" + testStatus + ".png");
    }
}
```

So, for `testScenario3()` the Android driver would launch the previous version of the app, as defined in the `appOld` parameter for Android in `uitests.properties`. Similarly, you can introduce a parameter for iOS to implement the same scenario. Now, let's see how to test multiple apps in the test suite.

Testing Multiple Apps in Same Test Suite

Suppose you want to test a particular scenario in a different app using the same test suite. The process is the same as running the test suite for a different version of the same app. We just have to change both app and appPackage and recreate the driver. You can create a new parameter app2 in uitests.properties. So, if the appPackage for app2 is "xxxx" and you are running testScenario4() in app2, then the method will need to be updated as shown in Listing 14-2.

Listing 14-2. Updated testScenario4 in AboutAppTestSuite.java

```
@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the App Version is displayed")
@Test
@Order(4)
@SIT
public void testScenario4() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    String app = getCustomProperties().get("app2");
    session.getAppiumDriver().quit();
    getCustomProperties().put("appPackage", "xxxx");
    getCustomProperties().put("app", app);
    try {
        session = startDefaultSession();
    } catch (Exception e) {
        e.printStackTrace();
    }
```

```
    try {
        SoftAssertions.assertSoftly(
                softAssertions -> {
                    softAssertions.assertThat(aboutAppScreen.
                    isAppVersionDisplayed("xxxx")).as("The App
                    Version is displayed").isTrue();
                }
        );
    } finally {
        testStatus = aboutAppScreen.isAppVersionDisplayed
        ("xxxx") ? "Passed" : "Failed";
        updateTCPassCount();
        try {
            aboutAppScreen.takeScreenShot("test-result/
            screenshots/" + tcName + "--" + testStatus + ".png");
        } catch (Exception e) {
            e.printStackTrace();
        }
        imageList.add("test-output/ScreenShots/" + tcName +
        "--" + testStatus + ".png");
    }
}
```

Summary

In this chapter, we learned how to test multiple apps and multiple versions of the same app in the same test suite. As I mentioned in the previous section, this is not a good practice unless there is a good use case, like testing app-upgrade scenarios. In the next chapter, we will learn how to run scripts or batch files from our test suite.

CHAPTER 15

Running Scripts or Batch Files from Test Suite

In the previous chapter, we learned how to test multiple apps and versions of same app from a single test suite. In this chapter, we will see how to run script or batch files from our test suite, along with a few examples.

Scenarios in Which Running a Script or Batch Files Is Required

You will recall that in Chapter 5, while creating the Appium `DefaultService`, we ran a script named `command.txt`. In real life, we may find it useful to use such scripts. For example, suppose we are verifying device logs as part of doing an assertion. For iOS, we store Appium and device logs together in a `log.txt` file. However, for Android we need to use adb commands. We can use scripts (or batch files for Windows) for this purpose.

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_15

Running Script or Batch Files from Test Suite

When we want to run a script from within our test suite, we need to create and call a method that in turn executes this script. Then, we need to create and call another method that does the verification and returns a Boolean value based on the verification outcome. We need to feed this second method to our `assert` statement. The remainder of this section provides a few examples.

We will implement the Android device log check in this chapter. You can implement the iOS device log check in a similar way. I will also show how to create and give permission to a script file during runtime.

First, we create a few script files to take the Android device log (last 5000 lines) through adb, clear the test-result PDF/screenshots, and assign executable permission to a script file, as shown in Listings 15-1 through 15-3.

Listing 15-1. adb_log_command.txt

```
#! /bin/bash
cd ~/Downloads/testautomation/appautomation
rm devicelog_android.txt
cd ~
./adb shell logcat -t 5000 > Downloads/testautomation/
appautomation/devicelog_android.txt
```

Listing 15-2. clear_testresult_command.txt

```
#! /bin/bash
cd ~/Downloads/testautomation/appautomation/test-result/
pdfreport
rm -rf *.pdf
cd ~/Downloads/testautomation/appautomation/test-result/
screenshots
rm -rf *.png
```

Listing 15-3. make_executable_command.txt

```
#! /bin/bash
cd ~/Downloads/testautomation/appautomation
chmod u+x xxxx.txt
```

In the `make_executable_command.txt` script (Listing 15-3), we assign executable permission to the `xxxx.txt` file created by the `createScriptFile(String command)` method in the `LogUtil` class. Please note the path in `Adb_log_command.txt` (Listing 15-1) is specific to our project path. Pay special attention to the path in `Clear_testresult_command.txt` (Listing 15-2), as otherwise the wrong content may get deleted.

Let's now create a `logutil` folder within the `src/main/java/com/taf/testautomation/utilities` folder. Next, we will create a `LogUtil` class, as shown in Listing 15-4.

Listing 15-4. LogUtil.java

```
package com.taf.testautomation.utilities.logutil;

import lombok.extern.slf4j.Slf4j;

import java.io.*;

import static com.taf.testautomation.utilities.excelutil.ExcelU
til.getCustomProperties;

@Slf4j
public class LogUtil {

    private static PrintStream ps;
    private static FileOutputStream fos = null;

    static {
        try {
```

```
            ps = new PrintStream(new File("./output.txt"));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }

    public LogUtil() {
    }

    public static void logOutput(String content) {
        PrintStream console = System.out;
        try {
            System.setOut(ps);
            System.out.append(content);
        } catch (Exception e) {
            log.info(e.getMessage());
        }
        System.setOut(console);
    }

    public static void takeDeviceLog() {
        try {
            Runtime.getRuntime().exec("./adb_log_command.txt");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void clearTestOutPut() {
        try {
            Runtime.getRuntime().exec("./clear_testresult_
            command.txt");
        } catch (IOException e) {
            e.printStackTrace();
```

```
    }
}

public static void executeCommandFile() {
    try {
        Runtime.getRuntime().exec("./make_executable_
        command.txt");
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public static void createScriptFile(String command) {
    try {
        String commandFilePath = getCustomProperties().
        get("reportPrefix") + "xxxx.txt";
        File commandFile = new File(commandFilePath);
        BufferedWriter writer = new BufferedWriter
        (new FileWriter(commandFile));
        String temp = "";
        writer.write("#! /bin/bash");
        writer.newLine();
        switch (command) {
            case "command1":
                temp = "xxxx";
                log.info("the command is" + temp);
                break;
            case "command2":
                temp = "yyyy";
                log.info("the command is" + temp);
                break;
            default:
                temp = "zzzz";
```

```
                log.info("the command is" + temp);
                break;
            }
            writer.write(temp);
            writer.newLine();
            String lastLine = "xxxx";
            writer.write(lastLine);
            writer.flush();
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static boolean deviceLogCheck(String content) {
        if (content.contains("xxxx")) {
            content = "xxxx";
        } else {
            content = "xxxx";
        }
        try {
            String filePath = getCustomProperties().
            get("reportPrefix") + "devicelog_android.txt";
            BufferedReader br = new BufferedReader
            (new FileReader(filePath));
            String strLine = null;
            while ((strLine = br.readLine()) != null) {// read
            every line in the file
                if (strLine.contains(content)) {
                    return true;
                }
            }
```

```
            br.close();
        } catch (Exception e1) {
            log.info(e1.getMessage());
        }
        return false;
    }
}
```

The logOutput(String content) method redirects the output of System.out to the output.txt file. This is useful if the user wants to log something during test execution. For example, suppose the user wants to print the appVersion label in the output.txt file. For this, the user needs to update the isAppVersionDisplayed(String text) method of the page object, as shown in Listing 15-5.

Listing 15-5. isAppVersionDisplayed Method in AboutAppScreen.java

```
/**
 * Validate if App Version is displayed
 *
 * @return True if App Version is displayed , otherwise returns
 * false.
 */
@Step("Verifying the App Version is Displayed")
public boolean isAppVersionDisplayed(String text) {
    IntStream.range(0, 2).forEach(i -> scrollUpTouchAction());
    waitInSeconds(MIN_WAIT);
    LogUtil.logOutput("App Version label on Screen " + "is: " +
    appVersion.getText() + "\n");
    return doesElementExist(appVersion, SMALL_WAIT) &&
    appVersion.getText().contains(text);
}
```

The next three methods are self-explanatory. The `createScriptFile(String command)` method creates a script runtime with the command parameter and saves it as `xxxx.txt`. The method `deviceLogCheck(String content)` checks the device log for any given string and returns a Boolean value based on whether the string is found in the log or not. This is the method that we will use in our test suite.

Listing 15-6 shows the updated `testScenario2()` in our test suite, with the new code added to check Android device log. You can similarly implement an iOS device log check.

Listing 15-6. Updated testScenario2() Method in AboutAppTestSuite.java

```
@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the App Logo is displayed")
@Test
@Order(2)
@Smoke
public void testScenario2() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    LogUtil.takeDeviceLog();
    try {
        SoftAssertions.assertSoftly(
                softAssertions -> {
                    softAssertions.assertThat(aboutAppScreen.
                    isAppLogoDisplayed()).as("The App Logo is
                    displayed").isTrue();
                }
```

```
        );
        SoftAssertions.assertSoftly(
                softAssertions -> {
                    softAssertions.assertThat(LogUtil.deviceLog
                    Check("xxxx")).as("Device log shows correct
                    data").isTrue();
                }
        );
    } finally {
        testStatus = aboutAppScreen.isAppLogoDisplayed() ?
        "Passed" : "Failed";
        updateTCPassCount();
        try {
            aboutAppScreen.takeScreenShot("test-result/
            screenshots/" + tcName + "--" + testStatus +
            ".png");
        } catch (Exception e) {
            e.printStackTrace();
        }
        imageList.add("test-output/ScreenShots/" + tcName +
        "--" + testStatus + ".png");
    }
}
```

Summary

In this chapter, we learned how to create and use script or batch files in our test suite. In the next chapter, we will discuss API testing using WebClient. It is from Spring library.

CHAPTER 16

API Testing

In the previous chapter, we learned how to run script or batch files from our test suite and looked at a few examples. In this chapter, I will show how, as an architect, you can design API tests. Let's get started.

Testing REST API with Web Client

I have implemented the rest calls here using a web client API, as mentioned in Chapter 1. However, you can use any other approach, like the RestAssured, HttpURLConnection, or Unirest APIs. In HP QC integration, I have used the Unirest API.

Let's create a class named `RestServices`, as shown in Listing 16-1, within the `com/taf/testautomation/services` folder. I have implemented `GET` and `POST REST` calls here. The `GET` call is used to fetch a resource from the server represented by the endpoint. The `POST` call is used to create a new resource on the server. You can similarly implement `PUT` (used to update a resource on the server) and `DELETE` (used to delete a resource from a server) calls.

Listing 16-1. RestServices.java

```
package com.taf.testautomation.services;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
```


K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_16

```
import com.google.gson.JsonElement;
import com.google.gson.JsonParser;
import org.springframework.http.MediaType;
import org.springframework.web.reactive.function.BodyInserters;
import org.springframework.web.reactive.function.client.
WebClient;
import reactor.core.publisher.Mono;

import java.io.FileWriter;
import java.io.Writer;

public class RestServices {

    public static JsonElement getJson(String url, String
    userName, String password, String saveAsPath) throws
    Exception {
        WebClient webClient = getWebClient(url);
        String responseString = webClient.get().headers
        (header -> header.setBasicAuth(userName, password)).
        accept(MediaType.APPLICATION_JSON).retrieve().
        bodyToMono(String.class).block();

        JsonParser jsonParser = new JsonParser();
        JsonElement response = jsonParser.
        parse(responseString);

        saveJSON(saveAsPath, response);

        return response;
    }

    public static WebClient getWebClient(String url) {
        return WebClient.builder().baseUrl(url).build();
    }
```

```
    public static void saveJSON(String path, JsonElement
    response) throws Exception {
        try (Writer writer = new FileWriter(path)) {
            Gson gson = new GsonBuilder().setPrettyPrinting().
            serializeNulls().create();
            gson.toJson(response, writer);
        }
    }

    public static JsonElement postJson(String url, JsonElement
    body, String appKeyHeader, String appKey, String
    saveAsPath) {
        Gson gson = new Gson();
        String requestJson = gson.toJson(body);
        return httpPost(url, appKeyHeader, appKey,
        requestJson);
    }

    public static JsonElement httpPost(String url, String
    appKeyHeader, String appKey, String requestJson) {
        WebClient webClient=WebClient.create();
        String postString = webClient.post().uri(url)
        .contentType(MediaType.APPLICATION_JSON)
        .header(appKeyHeader,appKey).accept
        (MediaType.APPLICATION_JSON).body(BodyInserters.fro
        mPublisher(Mono.just(requestJson), String.class)).
        retrieve().bodyToMono(String.class).block();

        JsonParser jsonParser = new JsonParser();
        JsonElement response = jsonParser.parse(postString);

        return response;
    }

}
```

The `JsonElement` type comes from the `Gson` library from Google, which we integrated in `build.gradle`. Normally, for authentication in REST, either a username/password combination or appkey is required. I have provided one example for each. The `getWebClient(String url)` method uses the builder pattern to implement the rest call. You can opt to use the factory pattern if needed. The other way of creating a web-client instance is to use the static `create()` method in the `WebClient` class.

The web client object is then used to call various REST methods that the `WebClient` interface supports. The `bodyToMono(String.class)` method is used to format the content of the API call. Here, since we are using `String`, we need to have the response string parsed to `JsonElement` using a `JsonParser` object, which we did in the `getJson(String url, String userName, String password, String saveAsPath)` and `httpPost(String url, String appKeyHeader, String appKey, String requestJson)` methods.

Updating TestautomationApplicationTests

Now, let's update the `TestautomationApplicationTests` class, discussed in Chapter 2, as shown in Figure 16-1.

```
package com.taf.testautomation.apitests;

import com.google.gson.JsonElement;
import com.taf.testautomation.TestAutomationProperties;
import com.taf.testautomation.services.RestServices;
import lombok.extern.slf4j.Slf4j;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit.jupiter.SpringExtension;

@Slf4j
@ExtendWith(SpringExtension.class)
@SpringBootTest
class TestautomationApplicationTests {

    @Autowired
    TestAutomationProperties testAutomationProperties;

    @Test
    public void testGetServiceCall() throws Exception {
        JsonElement response = RestServices.getJson(testAutomationProperties.getUrl(), "", "",
"sample-service.json");
        log.info(String.valueOf(response));

        if (response.isJsonNull()) {
            Assertions.fail();
        }
    }
}
```

Figure 16-1. *Updated TestautomationApplicationTests class*

The `@Autowired` annotation is from Spring, and it allows us to inject the `TestAutomationProperties` (created in Chapter 4) object dependency. This is possible because the `TestAutomationProperties` class is configured as a bean with an `@Configuration` Spring annotation. We have also set the `@Data` annotation in the `TestAutomationProperties` class, so we can now access the variable URL by just calling `testAutomationProperties.getUrl()`. This again demonstrates the power of Lombok and Spring.

Once we have the response object, we can validate its content in the way we want. In the preceding code, I verify for the `not null` condition. I leave it to you to validate the content of `JsonElement`.

Summary

In this chapter, we learned how to design an API test suite. In the next chapter, we will write utility methods to automate various device management functions, like toggling Wi-Fi, settings device date, time, time zone, and so on.

CHAPTER 17

Advanced Topic 1: Adding Device Management Functions

In the previous chapter, we learned how to design and write API tests, including examples for `GET` and `POST REST` calls. In this chapter, we will look at some ways to interact with device apps via the device settings. I have categorized this chapter as an advanced topic, and you can skip it for the time being. However, it will be required reading once you start interacting with device settings in the test suite.

Overview

We start by creating an `AppiumUtil` class, as shown in Listing 17-1, within the `com/taf/testautomation/devicemanagement` folder. We then implement methods for unlocking the device, toggling Wi-Fi, changing language, altering date settings, and enabling and disabling app notifications. We also implement the method

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_17

`getDeviceProperties(String devProp)`, which is used in our test suite to gather device data. I have written these methods for my Google Pixel and iPhone X phones. You may have to update these methods based on your phone model. Let's examine the code for `AppiumUtil`, and then I will explain it.

Listing 17-1. AppiumUtil.java

```
package com.taf.testautomation.devicemanagement;

import com.google.common.collect.ImmutableMap;
import io.appium.java_client.AppiumDriver;
import io.appium.java_client.MobileElement;
import io.appium.java_client.TouchAction;
import io.appium.java_client.touch.offset.PointOption;
import lombok.extern.slf4j.Slf4j;
import org.openqa.selenium.Dimension;

import java.time.Duration;
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
import java.util.*;
import java.util.concurrent.TimeUnit;
import java.util.stream.IntStream;

import static com.taf.testautomation.utilities.excelutil.ExcelU
til.getCustomProperties;
import static io.appium.java_client.touch.
WaitOptions.waitOptions;
import static java.time.Duration.ofMillis;

@Slf4j
public class AppiumUtil {
```

```
private AppiumDriver mobileDriver;
private Map<String, Object> targetMap;
private String hour = "", minute = "", targetYear = "",
targetMonth = "", targetDate = "", XPATH_IOS = "";
private String XPATH_YEAR_PICKER_CURRENT = "", XPATH_
YEAR_PICKER_TARGET = "", XPATH_DATE = "", XPATH_HOUR = "",
XPATH_MINUTE = "";
private static int i = 0;
private static final String XPATH_LOCALE_DEFAULT =
"//*[contains(@text,'Use locale default')]";
private static final String XPATH_24HOUR_FORMAT =
"//*[contains(@text,'Use 24-hour format')]";
private static final String XPATH_NETWORK_TIME_TOGGLE =
"//*[contains(@text,'Use network-provided time')]";
private static final String XPATH_TIME_SET =
"//*[contains(@text,'Time')]";
private static final String XPATH_DATE_SET =
"//*[@text='Date']";
private static final String XPATH_YEAR = "//*[@resource-
id='android:id/date_picker_header_year']";
private static final String XPATH_PREV_MONTH =
"//*[@content-desc='Previous month']";
private static final String XPATH_CLOCK_OK =
"//*[contains(@resource-id,'android:id/button1')]";
private static final String XPATH_LANGUAGE_CURRENT =
"//*[contains(@content-desc,'1,')]/android.widget.
ImageView";
private static final String XPATH_LANGUAGE_ENG =
"//*[contains(@content-desc,'English') or contains
(@content-desc,'Englisch') or contains(@content-
desc,'Inglés')]/android.widget.ImageView";
```

```
    private static final String XPATH_LANGUAGE_DE =
    "//*[contains(@content-desc,'German') or contains(@content-
    desc,'Deutsch') or contains(@content-desc,'Alemán')]/
    android.widget.ImageView";
    private static final String XPATH_LANGUAGE_ES =
    "//*[contains(@content-desc,'Spanish') or contains
    (@content-desc,'Spanisch') or contains(@content-
    desc,'Español')]/android.widget.ImageView";
    private static final String XPATH_WIFI_ICON_IOS =
    "//XCUIElementTypeStaticText[@name='Wi-Fi']";
    private static final String XPATH_WIFI_TOGGLE_IOS =
    "//XCUIElementTypeSwitch[@name='Wi-Fi']";
    private static final String XPATH_SETTINGS_ICON_IOS =
    "//XCUIElementTypeButton[@name='Settings']";
    private static final String XPATH_GENERAL_MENU_IOS =
    "//XCUIElementTypeStaticText[@name='General']";
    private static final String XPATH_NOTIFICATIONS_MENU_IOS =
    "//XCUIElementTypeStaticText[@name='Notifications']";
    private static final String XPATH_ALLOW_NOTIFICATIONS_
    TOGGLE_IOS = "//XCUIElementTypeSwitch[@name='Allow
    Notifications']";
    private static final String XPATH_TIME_FORMAT_TOGGLE_IOS =
    "//XCUIElementTypeSwitch[@name='24-Hour Time']";
    private static final String XPATH_AUTOMATIC_TIME_TOGGLE_IOS
    = "//XCUIElementTypeSwitch[@name='Set Automatically']";
    private static final String XPATH_ABOUT_MENU_IOS =
    "//XCUIElementTypeStaticText[@name='About']";
    private static final String XPATH_HOME_MENU_IOS =
    "//XCUIElementTypeStaticText[@name='Home']";
    private static final String XPATH_DATE_TIME_MENU_IOS =
    "//XCUIElementTypeStaticText[@name='Date & Time']";
```

```
private static final String XPATH_BACK_TO_GENERAL_IOS =
"//XCUIElementTypeButton[@name='General']";
private static final String XPATH_BACK_TO_NOTIFICATIONS_IOS
= "//XCUIElementTypeButton[@name='Notifications']";
private static final String XPATH_BACK_TO_SETTINGS_IOS =
"//XCUIElementTypeButton[@name='Settings']";
private static final String XPATH_DATE_PICKER_WHEEL_IOS =
"//XCUIElementTypePickerWheel[@value='Today']";
private static final String XPATH_HOUR_PICKER_WHEEL_IOS =
"//XCUIElementTypePickerWheel[contains(@value,'clock')]";
private static final String XPATH_MINUTE_PICKER_WHEEL_IOS =
"//XCUIElementTypePickerWheel[contains(@value,'minutes')]";
private static final String XPATH_TIME_ZONE_CELL_IOS = "//
XCUIElementTypeCell[@name='Time Zone']";
private static final String XPATH_TIME_ZONE_RESULT_IOS =
"//XCUIElementTypeStaticText[contains(@name,'U.S.A')]";
private final String serialNumber = "Serial Number";
private final String modelName = "Model Name";
private final String deviceName = "Name";
private final String timeZone = "Time Zone";

public AppiumUtil(AppiumDriver<MobileElement> driver) {
    this.mobileDriver = driver;
    this.targetMap = new HashMap<String, Object>();
}

public void unlockDevice() {
    if (getCustomProperties().get("isAndroid").equals
    ("true")) {
        List<String> enterPinArgs = Arrays.asList(
                "text",
                "6509"
```

```
            );
            targetMap = ImmutableMap.of(
                    "command", "input",
                    "args", enterPinArgs
            );
            mobileDriver.executeScript("mobile: shell",
            targetMap);

            List<String> unlockPhoneArgs = Arrays.asList(
                    "keyevent",
                    "66"
            );

            targetMap = ImmutableMap.of(
                    "command", "input",
                    "args", unlockPhoneArgs
            );
            mobileDriver.executeScript("mobile: shell",
            targetMap);
        } else {
            log.info("do nothing");
        }
    }

    public void toggleWifiOff() {
        if (getCustomProperties().get("isAndroid").equals
        ("true")) {
            List<String> wifiOffArgs = Arrays.asList(
                    "broadcast",
                    "-a",
                    "io.appium.settings.wifi",
                    "--es",
                    "setstatus",
```

```
                "disable"
        );

        targetMap = ImmutableMap.of(
                "command", "am",
                "args", wifiOffArgs
        );
        mobileDriver.executeScript("mobile: shell",
        targetMap);
    } else {
        String bundleId = getCustomProperties().get
        ("iosBundleId");
        iosToggleWifi(bundleId);
    }
}

public void toggleWifiOn() {
    if (getCustomProperties().get("isAndroid").equals
    ("true")) {
        List<String> wifiOnArgs = Arrays.asList(
                "broadcast",
                "-a",
                "io.appium.settings.wifi",
                "--es",
                "setstatus",
                "enable"
        );

        targetMap = ImmutableMap.of(
                "command", "am",
                "args", wifiOnArgs
        );
```

```
            mobileDriver.executeScript("mobile: shell",
            targetMap);
        } else {
            String bundleId = getCustomProperties().get
            ("iosBundleId");
            iosToggleWifi(bundleId);
        }
    }

    private void iosToggleWifi(String bundleId) {
        mobileDriver.activateApp("com.apple.Preferences");
        MobileElement wifiIcon = (MobileElement) mobileDriver.
        findElementByXPath(XPATH_WIFI_ICON_IOS);
        Map<String, Object> tapWifiArgs = new HashMap<>();
        tapWifiArgs.put("element", wifiIcon.getId());
        tapWifiArgs.put("x", 2);
        tapWifiArgs.put("y", 2);
        mobileDriver.executeScript("mobile: tap", tapWifiArgs);
        MobileElement wifiToggle = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_WIFI_TOGGLE_IOS);
        Map<String, Object> tapWifiToggleArgs = new HashMap<>();
        tapWifiToggleArgs.put("element", wifiToggle.getId());
        tapWifiToggleArgs.put("x", 0);
        tapWifiToggleArgs.put("y", 0);
        mobileDriver.executeScript("mobile: tap", tapWifi
        ToggleArgs);
        MobileElement settingsLink = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_SETTINGS_
        ICON_IOS);
        Map<String, Object> settingsIconArgs = new HashMap<>();
        settingsIconArgs.put("element", settingsLink.getId());
        settingsIconArgs.put("x", 2);
```

```
    settingsIconArgs.put("y", 2);
    mobileDriver.executeScript("mobile: tap", settings
    IconArgs);
    mobileDriver.executeScript("mobile: tap",
    settingsIconArgs);
    mobileDriver.activateApp(bundleId);
}

public void changeLanguageSettings(String deviceLanguage) {
    List<String> languageArgs = Arrays.asList(
            "start",
            "-a",
            "android.settings.LOCALE_SETTINGS"
    );

    targetMap = ImmutableMap.of(
            "command", "am",
            "args", languageArgs
    );
    mobileDriver.executeScript("mobile: shell", targetMap);
    MobileElement currentLanguage = (MobileElement)
    mobileDriver.findElementByXPath(XPATH_LANGUAGE_CURRENT);
    MobileElement desiredLanguage = getDesiredLangElement
    (deviceLanguage);
    Dimension size = mobileDriver.manage().window().getSize();
    int width = (int) (size.getWidth() * 0.95);
    int start = desiredLanguage.getLocation().getY() +
    desiredLanguage.getSize().getHeight() / 2;
    int end = (int) (currentLanguage.getLocation().getY() +
    currentLanguage.getSize().getHeight() * 0.25);

    new TouchAction(mobileDriver).press(PointOption.point
    (width, start)).waitAction(waitOptions(ofMillis(1000
```

```
        ))).moveTo(PointOption.point(width, end)).release().
        perform();
        mobileDriver.manage().timeouts().implicitlyWait
        (20, TimeUnit.SECONDS);
        mobileDriver.activateApp(getCustomProperties().get
        ("appPackage"));
    }

    public void changeLanguageADB(String language, String
    country) {
        List<String> permissionArgs = Arrays.asList(
                "grant",
                "net.sanapeli.adbchangelanguage",
                "android.permission.CHANGE_CONFIGURATION"
        );

        targetMap = ImmutableMap.of(
                "command", "pm",
                "args", permissionArgs
        );
        mobileDriver.executeScript("mobile: shell", targetMap);

        List<String> changeLangArgs = Arrays.asList(
                "start",
                "-n",
                "net.sanapeli.adbchangelanguage/.AdbChange
                Language",
                "-e",
                "language",
                language,
                "-e",
                "country",
                country
        );
```

```
    targetMap = ImmutableMap.of(
            "command", "am",
            "args", changeLangArgs
    );
    mobileDriver.executeScript("mobile: shell", targetMap);
    mobileDriver.manage().timeouts().implicitlyWait(5,
    TimeUnit.SECONDS);
}

public void setDeviceDate(String date) {
    mobileDriver.runAppInBackground(Duration.ofMill
    is(10000));
    if (i == 0) {
        tapOn(XPATH_NETWORK_TIME_TOGGLE);
    }
    targetYear = date.substring(0, 4);
    targetMonth = date.substring(5, 7);
    targetDate = date.substring(8).
    replaceAll("^0+(?!$)", "");
    LocalDate currentDate = LocalDate.now();
    int currentYear = currentDate.getYear();
    int currentMonth = currentDate.getMonthValue();
    tapOn(XPATH_DATE_SET);
    tapOn(XPATH_YEAR);
    setLocatorsDatePicker(String.valueOf(currentYear),
    targetYear, targetDate);
    MobileElement yearText = (MobileElement) mobileDriver
    .findElementByXPath(XPATH_YEAR_PICKER_CURRENT);
    int x4Point = yearText.getLocation().getX() + yearText
    .getSize().getHeight() / 2;
    int y4Point = yearText.getLocation().getY() + yearText
    .getSize().getHeight() / 2;
```

```
        boolean yearFound = false;
        while (!yearFound) {
            new TouchAction(mobileDriver).press(PointOption.point
            (x4Point, y4Point)).waitAction(waitOptions(ofMillis
            (500))).moveTo(PointOption.point(x4Point, y4Point +
            200)).release().perform();
            try {
                tapOn(XPATH_YEAR_PICKER_TARGET);
                yearFound = true;
            } catch (Exception e) {
                e.getMessage();
                yearFound = false;
            }
        }
        MobileElement prevMonth = (MobileElement) mobileDriver
        .findElementByXPath(XPATH_PREV_MONTH);
        int x6Point = prevMonth.getLocation().getX() +
        prevMonth.getSize().getWidth() / 2;
        int y6Point = prevMonth.getLocation().getY() +
        prevMonth.getSize().getHeight() / 2;
        while (currentMonth > Integer.valueOf(targetMonth)) {
            new TouchAction(mobileDriver).press(PointOption
            .point(x6Point, y6Point)).waitAction(waitOptions
            (ofMillis(500))).release().perform();
            currentMonth--;
        }
        tapOn(XPATH_DATE);
        tapOn(XPATH_CLOCK_OK);
        mobileDriver.activateApp(getCustomProperties()
        .get("appPackage"));
    }
```

```
public void setDeviceTime(String time) {
    hour = time.substring(0, 2);
    minute = time.substring(3).equals("00") ? "0" :
    time.substring(3);
    setLocatorsHourMinute(hour, minute);
    tapOn(XPATH_NETWORK_TIME_TOGGLE);
    mobileDriver.manage().timeouts().implicitlyWait
    (2, TimeUnit.SECONDS);
    tapOn(XPATH_TIME_SET);
    tapOn(XPATH_HOUR);
    tapOn(XPATH_MINUTE);
    tapOn(XPATH_CLOCK_OK);
    i++;
    mobileDriver.activateApp(getCustomProperties()
    .get("appPackage"));
}

public void setDeviceTimeZone(String timeZone) {
    switch (timeZone) {
        case "PST":
            timeZone = "America/Los_Angeles";
            break;
        case "EST":
            timeZone = "America/New_York";
            break;
        default:
            timeZone = "America/Chicago";
            break;
    }
    List<String> timeZoneArgs = Arrays.asList(
            "call",
            "alarm",
```

```
                "3",
                "s16",
                timeZone
        );

        targetMap = ImmutableMap.of(
                "command", "service",
                "args", timeZoneArgs
        );
        mobileDriver.executeScript("mobile: shell", targetMap);
    }

    public void setDeviceTimeFormat(String format) {
        if (format.equals("24")) {
            mobileDriver.runAppInBackground(Duration.ofMill
            is(10000));
            openAndroidDateTimeSettings();
            tapOn(XPATH_LOCALE_DEFAULT);
            mobileDriver.manage().timeouts().implicitlyWait
            (2, TimeUnit.SECONDS);
            tapOn(XPATH_24HOUR_FORMAT);
        } else {
            mobileDriver.runAppInBackground(Duration.ofMill
            is(10000));
            openAndroidDateTimeSettings();
            tapOn(XPATH_LOCALE_DEFAULT);
        }
    }

    public void IosSetDeviceDate(String bundleId, String
    targetDate) {
        openIosDateTimeSettings();
```

```
MobileElement automaticTimeToggle = (MobileElement)
mobileDriver.findElementByXPath(XPATH_AUTOMATIC_TIME_
TOGGLE_IOS);
Map<String, Object> automaticTimeArgs = new HashMap<>();
automaticTimeArgs.put("element", automaticTimeToggle
.getId());
automaticTimeArgs.put("x", 2);
automaticTimeArgs.put("y", 2);
mobileDriver.executeScript("mobile: tap",
automaticTimeArgs);
String currentDate = new java.text.SimpleDateFormat
("MMM dd, yyyy").format(Calendar.getInstance().
getTime());
setLocatorStaticText(currentDate);
MobileElement dateLink = (MobileElement) mobileDriver
.findElementByXPath(XPATH_IOS);
Map<String, Object> dateArgs = new HashMap<>();
dateArgs.put("element", dateLink.getId());
dateArgs.put("x", 2);
dateArgs.put("y", 2);
mobileDriver.executeScript("mobile: tap", dateArgs);
currentDate = new java.text.SimpleDateFormat
("yyyy-MM-dd").format(Calendar.getInstance().
getTime());
LocalDate dateBefore = LocalDate.parse(targetDate);
LocalDate dateAfter = LocalDate.parse(currentDate);
long noOfDaysBetween = ChronoUnit.DAYS.between
(dateBefore, dateAfter);
int days = (int) noOfDaysBetween;
swipeOnWheel(XPATH_DATE_PICKER_WHEEL_IOS, days, "UP");
```

```
        mobileDriver.manage().timeouts().implicitlyWait
        (2, TimeUnit.SECONDS);
        MobileElement backToGeneral = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_BACK_TO_
        GENERAL_IOS);
        backToGeneral.click();
        MobileElement backToSettings = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_BACK_TO_
        SETTINGS_IOS);
        backToSettings.click();
        mobileDriver.activateApp(bundleId);
    }

    public void IosSetDeviceTime(String bundleId, String
    targetTime) {
        openIosDateTimeSettings();
        MobileElement automaticTimeToggle = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_AUTOMATIC_TIME_
        TOGGLE_IOS);
        Map<String, Object> automaticTimeArgs = new HashMap<>();
        automaticTimeArgs.put("element", automaticTimeToggle.
        getId());
        automaticTimeArgs.put("x", 2);
        automaticTimeArgs.put("y", 2);
        mobileDriver.executeScript("mobile: tap",
        automaticTimeArgs);
        String currentDate = new java.text.SimpleDateFormat
        ("MMM dd, yyyy").format(Calendar.getInstance()
        .getTime());
        setLocatorStaticText(currentDate);
        MobileElement dateLink = (MobileElement) mobileDriver.
        findElementByXPath(XPATH_IOS);
```

```
Map<String, Object> dateArgs = new HashMap<>();
dateArgs.put("element", dateLink.getId());
dateArgs.put("x", 2);
dateArgs.put("y", 2);
mobileDriver.executeScript("mobile: tap", dateArgs);
String currentTime = new java.
text.SimpleDateFormat("HH:mm:ss").
format(Calendar.getInstance().getTime());
int currentHour = Integer.parseInt(currentTime
.substring(0, 2));
int currentMinute = Integer.parseInt(currentTime
.substring(3, 5).equals("00") ? "0" : currentTime
.substring(3, 5));
int targetHour = Integer.parseInt(targetTime
.substring(0, 2));
int targetMinute = Integer.parseInt(targetTime
.substring(3).equals("00") ? "0" : targetTime
.substring(3));
int diffHour = Math.abs(currentHour - targetHour);
if (currentHour > targetHour) {
    swipeOnWheel(XPATH_HOUR_PICKER_WHEEL_IOS,
    diffHour, "UP");
} else {
    swipeOnWheel(XPATH_HOUR_PICKER_WHEEL_IOS, diffHour,
    "DOWN");
}
int diffMin = Math.abs(currentMinute - targetMinute);
if (currentMinute > targetMinute) {
    swipeOnWheel(XPATH_MINUTE_PICKER_WHEEL_IOS,
    diffMin, "UP");
} else {
```

```
            swipeOnWheel(XPATH_MINUTE_PICKER_WHEEL_IOS,
            diffMin, "DOWN");
        }
        mobileDriver.manage().timeouts().implicitlyWait(2,
        TimeUnit.SECONDS);
        MobileElement backToGeneral = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_BACK_TO_
        GENERAL_IOS);
        backToGeneral.click();
        MobileElement backToSettings = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_BACK_TO_
        SETTINGS_IOS);
        backToSettings.click();
        mobileDriver.activateApp(bundleId);
    }

    public void IosSetDeviceTimeZone(String bundleId, String
    targetTimeZone) {
        openIosDateTimeSettings();
        MobileElement automaticTimeToggle = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_AUTOMATIC_TIME_
        TOGGLE_IOS);
        Map<String, Object> automaticTimeArgs = new
        HashMap<>();
        automaticTimeArgs.put("element", automaticTimeToggle
        .getId());
        automaticTimeArgs.put("x", 2);
        automaticTimeArgs.put("y", 2);
        mobileDriver.executeScript("mobile: tap",
        automaticTimeArgs);
        MobileElement timeZoneCell = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_TIME_ZONE_
        CELL_IOS);
```

```
        String timeZoneValue = timeZoneCell.
        getAttribute("value");
        setLocatorStaticText(timeZone);
        MobileElement tZ = (MobileElement) mobileDriver
        .findElementByXPath(XPATH_IOS);
        tZ.click();
        MobileElement timeZoneSearch = (MobileElement)
        mobileDriver.findElementByXPath("//XCUIElementType
        SearchField[@name='" + timeZoneValue + "']");
        timeZoneSearch.sendKeys(targetTimeZone);
        MobileElement timeZoneResult = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_TIME_ZONE_
        RESULT_IOS);
        Map<String, Object> tZResultArgs = new HashMap<>();
        tZResultArgs.put("element", timeZoneResult.getId());
        tZResultArgs.put("x", 2);
        tZResultArgs.put("y", 2);
        mobileDriver.executeScript("mobile: tap", tZResultArgs);
        mobileDriver.manage().timeouts().implicitlyWait
        (2, TimeUnit.SECONDS);
        MobileElement backToGeneral = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_BACK_TO_
        GENERAL_IOS);
        backToGeneral.click();
        MobileElement backToSettings = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_BACK_TO_
        SETTINGS_IOS);
        backToSettings.click();
        mobileDriver.activateApp(bundleId);
    }
```

```
public void IosToggleAutomaticTimeZone(String bundleId) {
    openIosDateTimeSettings();
    MobileElement automaticTimeToggle = (MobileElement)
    mobileDriver.findElementByXPath(XPATH_AUTOMATIC_TIME_
    TOGGLE_IOS);
    Map<String, Object> automaticTimeArgs = new HashMap<>();
    automaticTimeArgs.put("element", automaticTimeToggle.
    getId());
    automaticTimeArgs.put("x", 2);
    automaticTimeArgs.put("y", 2);
    mobileDriver.executeScript("mobile: tap",
    automaticTimeArgs);
    mobileDriver.manage().timeouts().implicitlyWait
    (2, TimeUnit.SECONDS);
    MobileElement backToGeneral = (MobileElement)
    mobileDriver.findElementByXPath(XPATH_BACK_TO_
    GENERAL_IOS);
    backToGeneral.click();
    MobileElement backToSettings = (MobileElement)
    mobileDriver.findElementByXPath(XPATH_BACK_TO_
    SETTINGS_IOS);
    backToSettings.click();
    mobileDriver.activateApp(bundleId);
}

public void IosSetDeviceTimeFormat(String bundleId) {
    openIosDateTimeSettings();
    MobileElement timeFormatToggle = (MobileElement)
    mobileDriver.findElementByXPath(XPATH_TIME_FORMAT_
    TOGGLE_IOS);
```

```
    Map<String, Object> timeFormatArgs = new HashMap<>();
    timeFormatArgs.put("element", timeFormatToggle.getId());
    timeFormatArgs.put("x", 2);
    timeFormatArgs.put("y", 2);
    mobileDriver.executeScript("mobile: tap",
    timeFormatArgs);
    mobileDriver.manage().timeouts().implicitlyWait
    (2, TimeUnit.SECONDS);
    MobileElement backToGeneral = (MobileElement)
    mobileDriver.findElementByXPath(XPATH_BACK_TO_
    GENERAL_IOS);
    backToGeneral.click();
    MobileElement backToSettings = (MobileElement)
    mobileDriver.findElementByXPath(XPATH_BACK_TO_
    SETTINGS_IOS);
    backToSettings.click();
    mobileDriver.activateApp(bundleId);
}

public String getDeviceProperties(String devProp) {
    switch (devProp) {
        case "Name":
            if (getCustomProperties().get("isAndroid")
            .equals("true")) {
                devProp = getCustomProperties()
                .get("deviceName");
            } else {
                devProp = iosAboutMenu(getCustomProperties
                ().get("iosBundleId"), deviceName);
            }
            break;
```

```
case "Model":
    if (getCustomProperties().get("isAndroid")
    .equals("true")) {
        List<String> modelArgs = Arrays.asList(
                "ro.product.model"
        );

        targetMap = ImmutableMap.of(
                "command", "getprop",
                "args", modelArgs
        );
        devProp = String.valueOf(mobileDriver
        .executeScript("mobile: shell", targetMap));
    } else {
        devProp = iosAboutMenu(getCustomProperties
        ().get("iosBundleId"), modelName);
    }
    break;
case "OS":
    if (getCustomProperties().get("isAndroid")
    .equals("true")) {
        devProp = getCustomProperties().get
        ("platformName");
    } else {
        devProp = getCustomProperties().get
        ("iosPlatformName");
    }
    break;
case "Manufacturer":
    if (getCustomProperties().get
    ("isAndroid").equals("true")) {
        List<String> mfgArgs = Arrays.asList(
```

```
                "ro.product.manufacturer"
            );

            targetMap = ImmutableMap.of(
                "command", "getprop",
                "args", mfgArgs
            );
            devProp = String.valueOf(mobileDriver
            .executeScript("mobile: shell", targetMap));
        } else {
            devProp = "Apple";
        }
        break;
    case "Version":
        if (getCustomProperties().get("isAndroid")
        .equals("true")) {
            devProp = getCustomProperties()
            .get("platformVersion");
        } else {
            devProp = getCustomProperties()
            .get("iosPlatformVersion");
        }
        break;
    case "Serial_Number":
        if (getCustomProperties().get("isAndroid")
        .equals("true")) {
            List<String> snArgs = Arrays.asList(
                "ro.boot.serialno"
            );

            targetMap = ImmutableMap.of(
                "command", "getprop",
```

```
                    "args", snArgs
                );
                devProp = String.valueOf(mobileDriver.
                executeScript("mobile: shell", targetMap));
            } else {
                devProp = iosAboutMenu(getCustomProperties
                ().get("iosBundleId"), serialNumber);
            }
            break;
        case "TimeZone":
            Calendar now = Calendar.getInstance();
            TimeZone timeZone = now.getTimeZone();
            devProp = timeZone.getDisplayName();
            break;
        case "ReportDate":
            devProp = new java.text.SimpleDateFormat
            ("dd-MMM-yyyy").format(Calendar.getInstance().
            getTime());
            break;
        case "ReportTime":
            devProp = new java.text.SimpleDateFormat
            ("HH:mm:ss").format(Calendar.getInstance()
            .getTime());
            break;
        default:
            if (getCustomProperties().get("isAndroid")
            .equals("true")) {
                devProp = "Google 3 XL";
            } else {
                devProp = iosAboutMenu(getCustomProperties
                ().get("iosBundleId"), modelName);
```

```
            }
            break;
    }
    return devProp;
}

private MobileElement getDesiredLangElement(String
devLanguage) {
    MobileElement desiredLanguage;
    switch (devLanguage) {
        case "de-DE":
            desiredLanguage = (MobileElement) mobileDriver
            .findElementByXPath(XPATH_LANGUAGE_DE);
            break;
        case "en-US":
            desiredLanguage = (MobileElement) mobileDriver
            .findElementByXPath(XPATH_LANGUAGE_ENG);
            break;
        case "es-ES":
            desiredLanguage = (MobileElement) mobileDriver
            .findElementByXPath(XPATH_LANGUAGE_ES);
            break;
        default:
            desiredLanguage = (MobileElement) mobileDriver
            .findElementByXPath(XPATH_LANGUAGE_CURRENT);
            break;
    }
    return desiredLanguage;
}

private String iosAboutMenu(String bundleId, String locator) {
    mobileDriver.activateApp("com.apple.Preferences");
```

```
        MobileElement generalMenu = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_GENERAL_
        MENU_IOS);
        Map<String, Object> generalMenuArgs = new HashMap<>();
        generalMenuArgs.put("element", generalMenu.getId());
        generalMenuArgs.put("x", 2);
        generalMenuArgs.put("y", 2);
       mobileDriver.executeScript("mobile: tap", generalMenuArgs);
        MobileElement aboutMenu = (MobileElement) mobileDriver.
        findElementByXPath(XPATH_ABOUT_MENU_IOS);
        Map<String, Object> aboutMenuArgs = new HashMap<>();
        aboutMenuArgs.put("element", aboutMenu.getId());
        aboutMenuArgs.put("x", 2);
        aboutMenuArgs.put("y", 2);
       mobileDriver.executeScript("mobile: tap", aboutMenuArgs);
        mobileDriver.activateApp(bundleId);
        mobileDriver.activateApp("com.apple.Preferences");
        setLocatorText(locator);
        MobileElement targetElement = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_IOS);
        String target = targetElement.getAttribute("value");
        MobileElement backToGeneral = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_BACK_TO_
        GENERAL_IOS);
        backToGeneral.click();
        MobileElement backToSettings = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_BACK_TO_
        SETTINGS_IOS);
        backToSettings.click();
        mobileDriver.activateApp(bundleId);
        return target;
    }
```

```
public void disableAppNotification(String appPackage) {
    if (getCustomProperties().get("isAndroid").
    equals("true")) {
        List<String> disableNotificationArgs = Arrays.asList(
                "call",
                "notification",
                "10",
                "s16",
                appPackage,
                "i32",
                "11346",
                "i32",
                "0"
        );

        targetMap = ImmutableMap.of(
                "command", "service",
                "args", disableNotificationArgs
        );
       mobileDriver.executeScript("mobile: shell", targetMap);
    } else {
        String bundleId = getCustomProperties()
        .get("iosBundleId");
        String appName = getCustomProperties()
        .get("isAndroid").equals("true") ? "xxxx" : "yyyy";
        iOSToggleAppNotification(bundleId, appName);
    }
}

public void enableAppNotification(String appPackage) {
    if (getCustomProperties().get("isAndroid").
    equals("true")) {
```

```
            List<String> disableNotificationArgs = Arrays.asList(
                    "call",
                    "notification",
                    "10",
                    "s16",
                    appPackage,
                    "i32",
                    "11346",
                    "i32",
                    "1"
            );

            targetMap = ImmutableMap.of(
                    "command", "service",
                    "args", disableNotificationArgs
            );
            mobileDriver.executeScript("mobile: shell",
            targetMap);
        } else {
            String bundleId = getCustomProperties()
            .get("iosBundleId");
            String appName = getCustomProperties().
            get("isAndroid").equals("true") ? "xxxx" : "yyyy";
            iOSToggleAppNotification(bundleId, appName);
        }
    }

    public void iOSToggleAppNotification(String bundleId,
    String appName) {
        mobileDriver.activateApp("com.apple.Preferences");
        MobileElement notificationsMenu = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_NOTIFICATIONS_
        MENU_IOS);
```

```
Map<String, Object> notificationsMenuArgs = new
HashMap<>();
notificationsMenuArgs.put("element", notificationsMenu.
getId());
notificationsMenuArgs.put("x", 2);
notificationsMenuArgs.put("y", 2);
mobileDriver.executeScript("mobile: tap", notifications
MenuArgs);
setLocatorText(appName);
MobileElement homeText = (MobileElement) mobileDriver
.findElementByXPath(XPATH_HOME_MENU_IOS);
int x4Point = homeText.getLocation().getX() + homeText
.getSize().getWidth() / 2;
int y4Point = homeText.getLocation().getY() + homeText
.getSize().getHeight() / 2;
boolean appFound = false;
while (!appFound) {
    new TouchAction(mobileDriver).press(PointOption
    .point(x4Point, y4Point)).waitAction(waitOptions(
    ofMillis(500))).moveTo(PointOption.point(x4Point,
    y4Point - 200)).release().perform();
    try {
        tapOn(XPATH_IOS);
        appFound = true;
    } catch (Exception e) {
        e.getMessage();
        appFound = false;
    }
}
```

```
        MobileElement notificationToggle = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_ALLOW_
        NOTIFICATIONS_TOGGLE_IOS);
        Map<String, Object> notificationToggleArgs =
        new HashMap<>();
        notificationToggleArgs.put("element", notification
        Toggle.getId());
        notificationToggleArgs.put("x", 2);
        notificationToggleArgs.put("y", 2);
        mobileDriver.executeScript("mobile: tap", notification
        ToggleArgs);
        MobileElement backToNotifications = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_BACK_TO_
        NOTIFICATIONS_IOS);
        backToNotifications.click();
        MobileElement backToSettings = (MobileElement)
        mobileDriver.findElementByXPath(XPATH_BACK_TO_
        SETTINGS_IOS);
        backToSettings.click();
        mobileDriver.activateApp(bundleId);
    }

    private void openAndroidDateTimeSettings() {
        List<String> dateTimeArgs = Arrays.asList(
                "start",
                "-n",
                "com.android.settings/.Settings\\$DateTime
                SettingsActivity"
        );

        targetMap = ImmutableMap.of(
                "command", "am",
```

```
            "args", dateTimeArgs
    );
    mobileDriver.executeScript("mobile: shell", targetMap);
}

private void openIosDateTimeSettings() {
    mobileDriver.activateApp("com.apple.Preferences");
    MobileElement generalMenu = (MobileElement)
    mobileDriver.findElementByXPath(XPATH_GENERAL_
    MENU_IOS);
    Map<String, Object> generalMenuArgs = new HashMap<>();
    generalMenuArgs.put("element", generalMenu.getId());
    generalMenuArgs.put("x", 2);
    generalMenuArgs.put("y", 2);
    mobileDriver.executeScript("mobile: tap",
    generalMenuArgs);
    MobileElement dateTimeMenu = (MobileElement)
    mobileDriver.findElementByXPath(XPATH_DATE_TIME_
    MENU_IOS);
    Map<String, Object> dateTimeMenuArgs = new HashMap<>();
    dateTimeMenuArgs.put("element", dateTimeMenu.getId());
    dateTimeMenuArgs.put("x", 2);
    dateTimeMenuArgs.put("y", 2);
    mobileDriver.executeScript("mobile: tap",
    dateTimeMenuArgs);
}

private void setLocatorsHourMinute(String hour, String
minute) {
    XPATH_HOUR = "//android.widget.RadialTimePickerView.
    RadialPickerTouchHelper[@content-desc='" + hour + "']";
```

```
        XPATH_MINUTE = "//android.widget.RadialTimePickerView
        .RadialPickerTouchHelper[@content-desc='" +
        minute + "']";
    }

    private void setLocatorsDatePicker(String currentYear,
    String targetYear, String date) {
        XPATH_YEAR_PICKER_CURRENT = "//*[@text='" + currentYear
        + "' and @resource-id='android:id/text1']";
        XPATH_YEAR_PICKER_TARGET = "//*[@text='" + targetYear +
        "' and @resource-id='android:id/text1']";
        XPATH_DATE = "//*[@text='" + date + "']";
    }

    private void tapOn(String locator) {
        MobileElement element = (MobileElement) mobileDriver
        .findElementByXPath(locator);
        int xPoint = element.getLocation().getX() + element
        .getSize().getWidth() / 2;
        int yPoint = element.getLocation().getY() + element
        .getSize().getHeight() / 2;
        new TouchAction(mobileDriver).press
        (PointOption.point(xPoint, yPoint)).waitAction
        (waitOptions(ofMillis(500))).release().perform();
    }

    private void swipeOnWheel(String locator, int number,
    String direction) {
        MobileElement element = (MobileElement) mobileDriver
        .findElementByXPath(locator);
        int xPoint = element.getLocation().getX() + element
        .getSize().getWidth() / 2;
```

```
        int yPoint = element.getLocation().getY() + element
        .getSize().getHeight() / 2;
        if (direction.equals("UP")) {
            IntStream.range(0, number).forEach(i -> new
            TouchAction(mobileDriver).press
            (PointOption.point(xPoint, yPoint)).waitAction
            (waitOptions(ofMillis(500))).moveTo(PointOption
            .point(xPoint, yPoint + 40)).release().perform());
        } else {
            IntStream.range(0, number).forEach(i -> new
            TouchAction(mobileDriver).
            press(PointOption.point(xPoint, yPoint)).wait
            Action(waitOptions(ofMillis(500))).moveTo
            (PointOption.point(xPoint, yPoint - 40)).release()
            .perform());
        }
    }

    private void setLocatorText(String text) {
        XPATH_IOS = "//XCUIElementTypeCell[@name='" + text + "']";
    }

    private void setLocatorStaticText(String text) {
        XPATH_IOS = "//XCUIElementTypeStaticText[@name='" +
        text + "']";
    }
}
```

You can see that I have defined several locators for device settings. These locators are specific to my phone. They are all defined as final String and start with the XPATH prefix. The user interface elements for the first five locators (starting with XPATH_LOCALE_DEFAULT) can be seen in Figure 17-1. Figures 17-2, 17-3, and 17-4 show a few other elements. The remaining

locators represent various menu elements in the Settings screen. For example, `XPATH_HOME_MENU_IOS` represents the "Home" menu in the Settings screen of an iOS device, and so on. You should understand the method and the flow involved and modify that for your specific device.

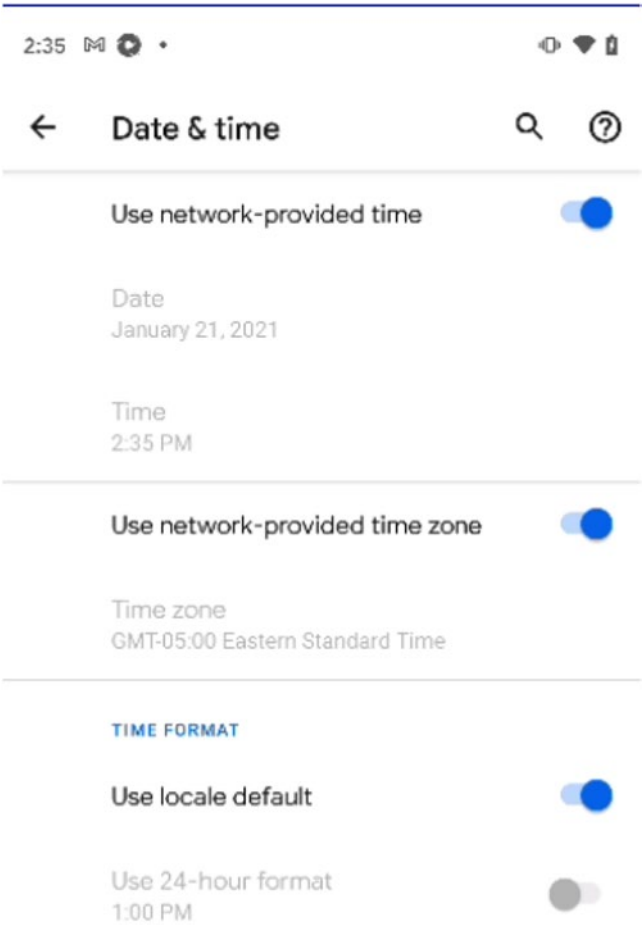


Figure 17-1. *Android Date & Time settings UI*

Figure 17-2. *Android date picker UI*

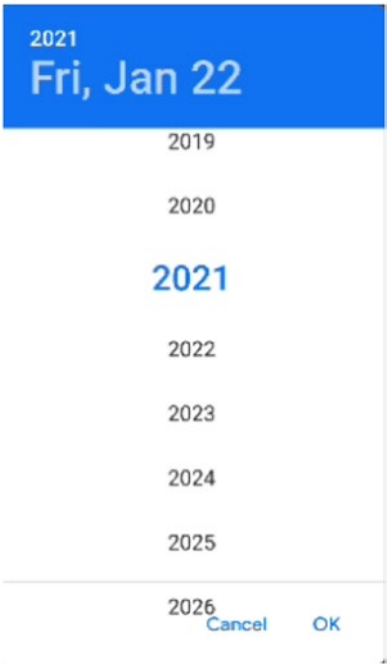


Figure 17-3. *Android year picker UI*

The string variables, which are initialized with "", will be assigned values in various methods. The last three variables except `timeZone` are device labels for serial number, model name, and device name, which can be seen in Settings under General ➤ About in iOS. The variable `timeZone` represents the label inside Date & Time in iOS. We need the `AppiumDriver` object for screen interaction and `targetMap` for running mobile shell commands, as described in the following sections. I have mostly used adb commands for Android and Appium commands for iOS.

Unlocking the Device

The first method is used to unlock the Android device. On my device, the unlock code is 6509. Key event 66 is used for tapping the button to enter the code. Adb is run through the `mobile:shell` command, which takes the input arguments in a `HashMap<String, Object>` data structure. The `mobile:shell` command is called through the `executeScript` method. There is plenty of online documentation available on the various adb commands. I urge you to look at such documentation to get to know the full capabilities of adb.

Unlike Android, it is not straightforward to unlock an iOS device through Appium. The swipe up to unlock is not automatable without hardcoding coordinates. I will avoid this topic here in this book so as to focus on architecture.

Toggling the Wi-Fi

The next two methods toggle the Wi-Fi in Android through adb. We take the same approach with different arguments. The method for iOS takes `bundleId` parameters. This is to ensure that after toggling the Wi-Fi we can navigate back to our application under testing. We use the `mobile:tap` command to tap various icons on the screen. The arguments for the `tap` command should be in a map, and `tap` is called through the `executeScript` method.

Setting the Language

To set the device language for Android, I have used an Appium-based method in addition to an adb method. This is to show that we can always use Appium to do the same job that adb does. However, adb makes it more convenient and reliable to carry out these changes. We also save lots of lines of code when using adb commands.

The `string` parameter in `changeLanguageSettings(String deviceLanguage)` should be in language–locale format. For example, `de-DE` means Germany, for Deutschland. Accordingly, the mobile element is returned by the `getDesiredLangElement(String devLanguage)` method. The limitation of this method is that we need to add all possible languages in Settings before running this code. On my Google Pixel, the screen navigation is Settings ➤ System ➤ Languages & Input ➤ Languages. The method then puts the desired language at the top of the list, making it the default, before navigating back to the application under testing.

The other method, `changeLanguageADB(String language, String country)` using adb, uses the freely distributed popular Android application `net.sanapeli.adbchangelanguage` to do the language change. So, you will need to install this application on your device prior to running this adb command. The first `mobile:shell` command in this method is used to give permission to this app, and the second `mobile:shell` command changes the device language. The language parameter should be a two-letter abbreviation of the desired language (example: `de` for German), while the country is an uppercase letter pair (example **DE** for Germany) representing the locale.

Setting the Device Date, Time, Time Zone, and Time Format

The next four methods are used to set device date, time, time zone, and time format in Android. Except for setting the device time zone, I have used Appium. The date parameter in `setDeviceDate(String date)` should be in YYYY-MM-DD format. I have also set text-based locators in various private methods. I defined a method `tapOn(String locator)` to tap on various locators. This is an alternative to using the `mobile:tap` command.

In the `setDeviceTimeZone(String timeZone)` method I have implemented code for three time zones. You will need to update the switch block to include more time zones in the code as necessary.

Please note that the Date & Time settings UI will be different for various phone models. On my device, the UI looks like Figure 17-1.

The `XPATH_YEAR` locator represents the year 2021, as in Figure 17-2.

The `XPATH_YEAR_PICKER_CURRENT` and `XPATH_YEAR_PICKER_TARGET` locators represent current and target years, as in Figure 17-3.

The next five methods are for iOS. There is an additional method in iOS named `IosToggleAutomaticTimeZone(String bundleId)` that can toggle the automatic time zone setting. The `bundleid` parameter is used to return to the application undergoing testing once the Settings changes are done. The Date & Time settings UI in my iOS looks like Figure 17-4.

Figure 17-4. *iOS Date & Time settings UI*

Reading Device Properties

The method `getDeviceProperties(String devProp)` implements the capability to read various device properties, like name, model, OS, manufacturer, OS version, device serial number, and time zone. In addition, we also generate the PDF report date and time from this method. You will recall that we called this method in our test suite to populate our PDF report.

Enabling and Disabling App Notifications

The last three public methods are used to enable and disable app notifications in Android and iOS. I have used adb commands in Android methods. The extra argument `appName` in the iOS method `iOSToggleAppNotification(String bundleId, String appName)` is used to set the locator for our application under test.

You can observe that in iOS methods, after making the Settings changes, we come back to the Settings root page. This is required, as in iOS once we open Settings and navigate to a screen—say, Date & Time—unless we return to the Settings root screen, the next time we open the Settings application we would see the last screen we were on.

Summary

In this chapter, we learned how to create various device management functions to interact with apps in the device Settings screen. In the next chapter, we will deal with a more advanced topic; i.e., how to integrate our framework with HP ALM.

CHAPTER 18

Advanced Topic 2: Integrating with HP ALM

In the previous chapter, we wrote several methods to interact with device settings. In this chapter, we will discuss a more advanced topic; i.e., how to integrate a framework with HP ALM. This is a value-add chapter for advanced readers and can be skipped if you are not using QC as your test management tool. Let's first discuss the ALM APIs that we can use.

Using ALM 15.x API

HP ALM exposes several utility infrastructure classes and application classes that you can use to read, create, update, or delete entities in HP QC. In this book, I use QC and ALM interchangeably. The infrastructure classes can be found at `https://admhelp.microfocus.com/alm/api_refs/REST_TECH_PREVIEW/Content/REST_API_Tech_Preview/General/Example_Application.html` in ALM 15.x. ALM being a licensed software, I cannot reproduce these classes here, and you'd need to have access to HP QC. The screenshot for the preceding link looks like Figure 18-1.

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_18

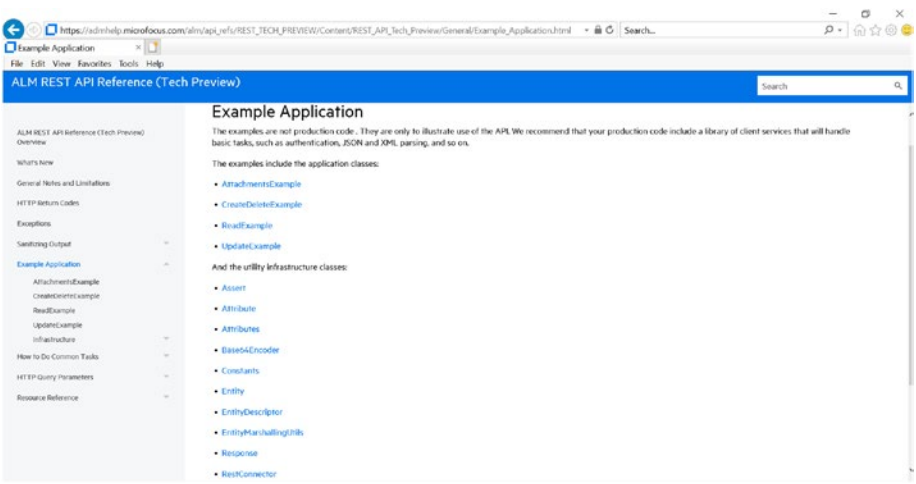


Figure 18-1. *ALM REST API reference page*

We will not need the `Attribute` and `Attributes` classes. However, we will use the other infrastructure classes. Let me briefly explain these.

`Assert` – This is used in assertions to verify CRUD operations. You will see this in application classes.

`Base64Encoder` – This is used to encode login credentials, namely username and password, in Base64 encoding before transmitting to authentication end point.

`Constants` – This is used to define variables like QC host, port, domain, project, and even login credentials. We also define our entity XML here for create/update operations.

`Entity` – In QC, *entity* means an XML Object with a set of fields. These represent components like test run, defect, requirement, etc.

`EntityDescriptor` - This specifies metadata for entities.

`EntityMarshallingUtils` – This is used to convert (`Marshall`) XML into Java objects and vice versa (`UnMarshall`).

`Response` – This captures the HTTP response.

`RestConnector` – This defines the HTTP methods supported by the ALM REST API, namely `GET`, `PUT`, `POST`, and `DELETE`.

We will add two methods, as shown in Listing 18-1, to the `RestConnector` class to update the cookies before a CRU operation from the test suite. I have seen that without these methods, CRU operations often fail in ALM 15.x. You may try without these methods if you are using an earlier version of ALM.

Listing 18-1. Methods to Update Cookies

```
public void getQCSession() {
    try {
        HttpResponse<JsonNode> response =
        Unirest.post(buildUrl("rest/site-session"))
                .header("Accept", "application/xml")
                .header("Content-Type", "text/html")
                .header("Cookie", getCookieString()).asJson();
        this.updateCookies(response);
        System.out.println("Post QC Session code is " +
        response.getStatus());
```

```
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void updateCookies(HttpResponse<JsonNode> response) {

    Iterable<String> newCookies =
            response.getHeaders().get("Set-Cookie");
    if (newCookies != null) {

        for (String cookie : newCookies) {
            int equalIndex = cookie.indexOf('=');
            int semicolonIndex = cookie.indexOf(';');

            String cookieKey = cookie.substring(0, equalIndex);
            String cookieValue =
                    cookie.substring(equalIndex + 1,
                    semicolonIndex);

            cookies.put(cookieKey, cookieValue);
        }
    }
}
```

The application classes provide examples for carrying out CRUD operations using the ALM REST API. I saw that the `update(String entityUrl, String updatedEntityXml)` method in the `Update` class sometimes does not work in 15.x. So, if you find the same issue, I suggest you replace the provided method with the method shown in Listing 18-2.

Listing 18-2. Method to Update Entity

```
/**
     * @param entityUrl        to update
```

```
     * @param updatedEntityXml New entity description. Only
     lists updated fields.
     *                        Unmentioned fields will not change.
     * @return xml description of the entity on the serverside,
     after update.
     */
    public String update(String entityUrl, String
    updatedEntityXml) {
        Map<String, String> headers = new HashMap<String,
        String>();
        headers.put("Accept", "application/xml");
        headers.put("Content-Type", "application/xml");
        headers.put("Cookie", RestConnector.getInstance()
        .getCookieString());

        HttpResponse<JsonNode> putResponse
                = Unirest.put(entityUrl)
                .headers(headers)
                .body(updatedEntityXml.getBytes())
                .asJson();
        System.out.println("Put response code is
        " +putResponse.getStatus());

        return putResponse.getStatusText();
}
```

In addition to the four application classes discussed, we can also write an `AuthenticateLoginLogout` class that provides authentication services to the other application classes. I have taken this class from https://github.com/alonso05/ALM/blob/master/src/qc/rest/examples/AuthenticateLoginLogoutExample.java and modified it a bit, as shown in Listing 18-3.

Listing 18-3. AuthenticateLoginLogout.java

```
package com.taf.testautomation.testmanagement.alm.application;

import com.taf.testautomation.testmanagement.alm.
infrastructure.Base64Encoder;
import com.taf.testautomation.testmanagement.alm.
infrastructure.Response;
import com.taf.testautomation.testmanagement.alm.
infrastructure.RestConnector;

import java.net.HttpURLConnection;
import java.util.HashMap;
import java.util.Map;

/**
 * This example shows how to login/logout/authenticate to the
server with REST.
 */
public class AuthenticateLoginLogout {

    private RestConnector con;

    /**
     * <p>
     * Once you have initialized the class RestConnector, you
     can use this
     * constructor to create a new object from AlmConnector
     since the referenced
     * class RestConnector is keeping the connection details.
     * </p>
     */
    public AuthenticateLoginLogout() {
        this.con = RestConnector.getInstance();
```

```
}

/**
 * For logging in to an ALM project. If a user is already
 * authenticated, True will be returned. If user is already
 authenticated with a different
 * credential, first log out before login
 *
 * @param username
 * @param password
 * @return true if user is successfully authenticated
 else false.
 * @throws Exception
 */
public boolean login(String username, String password)
throws Exception {
    /**
     * Get the current authentication status.
     */
    String authenticationPoint = this.isAuthenticated();
    if (authenticationPoint != null) {
        return this.login(authenticationPoint, username,
        password);
    }

    return true;
}

/**
 * Standard HTTP login (basic authentication), where
 * one must store the returned cookies for further use.
 *
 * @param loginUrl
```

```
     * @param username
     * @param password
     * @return true if login is successful, else false.
     * @throws Exception
     */
    private boolean login(String loginUrl, String username,
    String password)
            throws Exception {
        byte[] credBytes = (username + ":" + password).
        getBytes();
        String credEncodedString = "Basic " + Base64Encoder
        .encode(credBytes);

        Map<String, String> map = new HashMap<String,
        String>();
        map.put("Authorization", credEncodedString);

        Response response = con.httpGet(loginUrl, null, map);
        System.out.println("Login GET response is " + response
        .getStatusCode());

        boolean ret = response.getStatusCode() == HttpURL
        Connection.HTTP_OK;

        return ret;
    }

    /**
     * Closes a session and cleans session cookies on a client.
     *
     * @return true if logout successful.
     * @throws Exception
     */
```

```
public boolean logout() throws Exception {
    Response response = con.httpGet(
            con.buildUrl("authentication-point/logout"),
            null, null);
    return (response.getStatusCode() == HttpURL
    Connection.HTTP_OK);
}

/**
 * @return null if user already authenticated.
 *         else an URL to authenticate against.
 * @throws Exception
 *
 */
public String isAuthenticated() throws Exception {
    String isAuthenticateUrl = con.buildUrl("rest/
    is-authenticated");
    String ret;

    Response response = con.httpGet(isAuthenticateUrl,
    null, null);
    int responseCode = response.getStatusCode();

    /**
     * If already authenticated
     */
    if (responseCode == HttpURLConnection.HTTP_OK) {
        ret = null;
    }
    /**
     * If not authenticated yet, return a URL to
     * authenticate via www-authenticate.
     */
```

```
        else if (responseCode == HttpURLConnection.HTTP_
        UNAUTHORIZED) {
            ret = con.buildUrl("authentication-point/
            authenticate");
        }
        /**
         * Throws an Exception if an error occurred
         during login
         */
        else {
            throw response.getFailure();
        }

        return ret;
    }
}
```

We will now create another class named `AlmWorkflowUtil`, as shown in Listing 18-4, and write methods to log in to ALM, read, update, delete entities, and log out. We will use these methods from both our UI and API test suites to perform the desired actions.

Listing 18-4. AlmWorkflowUtil.java

```
package com.taf.testautomation.testmanagement.alm.application;

import com.taf.testautomation.testmanagement.alm.
infrastructure.*;
import com.taf.testautomation.utilities.emailutil.EmailUtil;

import java.io.File;
import java.io.FileInputStream;
import java.util.HashMap;
import java.util.List;
```

```
import java.util.Map;

import static com.taf.testautomation.utilities.excelutil.ExcelU
til.getCustomProperties;

public class AlmWorkflowUtil {

    AuthenticateLoginLogout alm = new Authenticate
    LoginLogout();
    RestConnector conn = RestConnector.getInstance();

    /**
     * Log in to HP ALM
     */
    public void almLogin(String userName, String password)
    throws Exception {
        conn.init(new HashMap<String, String>(),
        Constants.HOST,
                Constants.DOMAIN, Constants.PROJECT);
        alm.login(userName, password);
    }

    /**
     * Get the defect with given ID.
     */
    public void getAlmDefect(String defectID) throws
    Exception {
        String defectUrl = conn.buildEntityCollectionUrl(
        "defect");
        defectUrl += "/" + defectID;

        Map<String, String> requestHeaders = new
        HashMap<String, String>();
        requestHeaders.put("Accept", "application/xml");
```

```
        Response res = conn.httpGet(defectUrl, null,
        requestHeaders);

        // xml -> class instance
        String postedEntityReturnedXml = res.toString();
        Entity entity = EntityMarshallingUtils.marshal
        (Entity.class, postedEntityReturnedXml);

        /*
         * Print all names available in entity defect
         to screen.
         */
        List<Entity.Fields.Field> fields = entity.getFields()
        .getField();
        for (Entity.Fields.Field field : fields) {
            System.out.println(field.getName() + " : "
                    + field.getValue());
        }
    }

    /**
     * Get the testrun with given ID.
     */
    public void getAlmTestRun(String testRunID) throws
    Exception {
        conn.getQCSession();
        String testRunUrl = conn.buildEntityCollectionUrl("run");
        testRunUrl += "/" + testRunID;
        System.out.println("\n");
        System.out.println("\n");
        System.out.println("Test Run URL is" + testRunUrl);
```

```
    Map<String, String> map = new HashMap<String,
    String>();
    map.put("Accept", "application/xml");

    Response resp = conn.httpGet(testRunUrl, null, map);
    System.out.println("\n");
    System.out.println("\n");
    System.out.println(resp.getStatusCode());
}

/**
 * Create a new testrun in testlab from existing testset
 */
public void createAlmTestRun() throws Exception {
    conn.getQCSession();
    String requirementsUrl = conn.buildEntityCollectionUrl
    ("run");

    CreateDelete createTestRun = new CreateDelete();
    String newCreatedResourceUrl =
            createTestRun.createEntity(requirementsUrl,
                    Constants.entityToPostXmlTR);
}

/**
 * Update the testrun in testlab
 */
/**
 * Print all names available in entity testrun to screen.
 */
public void updateAlmTestRun(String tcStatus, String
updateKey, String updateValue) throws Exception {
    conn.getQCSession();
```

```
String updateURL = getCustomProperties()
.get("updateURL");
String id = "id";
String idValue = "";
// xml -> class instance
Entity testRun = EntityMarshallingUtils.marshal
(Entity.class,
        Constants.entityToPostXmlTR);
List<Entity.Fields.Field> fieldsTR = testRun
.getFields().getField();
for (
        Entity.Fields.Field field : fieldsTR) {
    System.out.println(field.getName() + " : "
            + field.getValue());
    if (field.getName().equals("id")) {
        String str = field.getValue().toString();
        idValue = str.substring(str.indexOf("[") + 1,
        str.indexOf("]"));
    }
}
updateURL = updateURL + idValue;
String name = updateKey;
String nameUpdateValue = updateValue;
System.out.println("String parameters are " + id + " "
+ idValue + " " + name + " " + nameUpdateValue);
Update update = new Update();
//create the update content
String updatedEntityXml =
        update.generateSingleFieldUpdateXmlTR(name,
                nameUpdateValue, id, idValue);
```

```
    //update the testrun
    String put = update.update(updateURL,
            updatedEntityXml);
}

/**
 * Update the testrun in testlab
 */
/**
 * Print all names available in entity testrun to screen.
 */
public void updateAlmTestRunWithAttachment(String tcStatus,
String filePath, String contentType) throws Exception {
    String updateURL = getCustomProperties()
    .get("updateURL");
    String id = "id";
    String idValue = "";
    // xml -> class instance
    Entity testRun = EntityMarshallingUtils.marshal
    (Entity.class, Constants.entityToPostXmlTR);
    List<Entity.Fields.Field> fieldsTR = testRun
    .getFields().getField();
    for (
            Entity.Fields.Field field : fieldsTR) {
        System.out.println(field.getName() + " : "
                + field.getValue());
        if (field.getName().equals("id")) {
            String str = field.getValue().toString();
            idValue = str.substring(str.indexOf("[") + 1,
            str.indexOf("]"));
        }
    }
```

```
        updateURL = updateURL + idValue;

        System.out.println("String parameters are " + id +
        idValue);

        //update the testrun
        String url = attachFile(updateURL, filePath,
        contentType);

        //Send emails to distribution list
        EmailUtil.sendEmailAttachment("Test run was updated
        with attachment after test execution - " + updateURL,
        tcStatus);
    }

    /**
     * Delete an existing testrun in testlab
     */
    public void deleteAlmTestRun() throws Exception {
        String deleteURL = getCustomProperties()
        .get("deleteURL");
        String id = "id";
        String idValue = "";
        // xml -> class instance
        Entity testRun = EntityMarshallingUtils.marshal
        (Entity.class, Constants.entityToPostXmlTR);
        List<Entity.Fields.Field> fieldsTR = testRun
        .getFields().getField();
        for (
                Entity.Fields.Field field : fieldsTR) {
            System.out.println(field.getName() + " : "
                    + field.getValue());
            if (field.getName().equals("id")) {
```

```
                String str = field.getValue().toString();
                idValue = str.substring(str.indexOf("[") + 1,
                str.indexOf("]"));
            }
        }
        deleteURL = deleteURL + idValue;

        CreateDelete deleteTestRun = new CreateDelete();
        String serverResponse =
                deleteTestRun.deleteEntity(deleteURL);
        System.out.println(serverResponse);
    }

    /**
     * Log out from HP ALM
     */
    public void almLogout() throws Exception {
        alm.logout();
    }

    private String attachFile(String updateUrl, String
    filePath, String contentType) throws Exception {
        File uploadFile = new File(filePath);
        String fileName = uploadFile.getName();
        FileInputStream inputStream = new
        FileInputStream(uploadFile);
        byte[] byteSteam = new byte[inputStream.available()];
        inputStream.read(byteSteam);
        String multipartFileDescription = "File description";

        Attachments attachments = new Attachments();

        String newImageAttachmentUrl =
                attachments.attachWithMultipart(
```

```
                        updateUrl,
                        byteSteam,
                        contentType,
                        fileName,
                        multipartFileDescription);
        return newImageAttachmentUrl;
    }
}
```

You can see that in the `AlmWorkflowUtil` class I have called the `getQCSession()` method inside the methods to read, create, and update the test run. There is also the static method `sendEmailAttachment` from the `EmailUtil` class that will give a compiler exception. So, I urge you to comment the method from the code now and uncomment it back after we implement this method in Appendix A.

Let's create two subfolders named `application` and `infrastructure` within the `com/taf/testautomation/testmanagement/alm` folder. Put the `application` and `infrastructure` classes in their corresponding folders. The folder structure should now look like Figure 18-2.

Figure 18-2. *Folder structure with ALM integration*

In the following sections, we will discuss how to perform login and authentication and CRUD operations from our test classes using the utility methods just described.

Login and Authentication

In both the `AboutAppTestSuite` and `TestautomationApplicationTests` classes, we have to call the `almLogin` method in the `AlmWorkflowUtil` class. This method first invokes the `RestConnector` method to initialize the `HOST`, `DOMAIN`, and `PROJECT` variables as defined in the `Constants` class. Then it calls the `login` method in the `AuthenticateLoginLogout` class with `username` and `password` as parameters.

CRUD Operations in AboutAppTestSuite

In `AboutAppTestSuite`, I update a test run after the test case has been executed. The `updateAlmTestRunWithAttachment(String tcStatus, String filePath, String contentType)` method also updates the test run with an attachment. The user can choose various file types as attachments, as supported in the `uitests.properties` file. I have also renamed the last method to the more meaningful `create_pdf_report_update_hpalm`. The last method from the test suite now looks like Listing 18-5.

Listing 18-5. Report Method in AboutAppTestSuite

```
@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Create Report and update HP ALM")
@Test
```

```
@Order(6)
@Smoke
@Regression
@SIT
@AT
public void create_pdf_report_update_hpalm() throws Exception {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    appiumUtil = new AppiumUtil(getSession()
    .getAppiumDriver());
    String pdfFile = getCustomProperties()
    .get("reportPrefix") + "test-result/pdfreport/" +
    TEST_NAME + "Report.pdf";
    String testCase = TEST_NAME + "Screenshots:";
    String timeZone = appiumUtil.getDeviceProperties
    ("TimeZone");
    String reportDate = appiumUtil.getDeviceProperties
    ("ReportDate");
    String reportTime = appiumUtil.getDeviceProperties
     ("ReportTime");
    String deviceName = appiumUtil.
    getDeviceProperties("Name");
    String deviceOS = appiumUtil.getDeviceProperties("OS");
    String deviceModel = appiumUtil.getDevice
    Properties("Model");
    String deviceMfg = appiumUtil.getDeviceProperties
    ("Manufacturer");
    String osVersion = appiumUtil.getDeviceProperties
    ("Version");
```

```
    String deviceSN = appiumUtil.getDeviceProperties
    ("Serial_Number");
    String buildNumber = getCustomProperties()
    .get("build");
    Stream<String> propStream = Stream.of(testCase, timeZone,
    reportDate, reportTime, deviceName, deviceOS, deviceModel,
    deviceMfg, osVersion, deviceSN, buildNumber);
    List<String> propList = propStream.collect(Collectors.
    toList());

    String update = j + " of " + i + " Passed";
    new PdfUtil().getPdfFromImageList(propList, imageList, new
    File(pdfFile));
    PdfUtil.mergePdf(new File(getCustomProperties()
    .get("mergedReport")));
    updateALMEntity(update);
}

private void updateALMEntity(String tcStatus) throws
Exception {
    almWorkflowUtil.almLogin(getCustomProperties().
    get("almUsername"), getCustomProperties().
    get("almPassword"));
    switch (testStatus) {
        case "Passed":
            almWorkflowUtil.updateAlmTestRun(tcStatus,
            getCustomProperties().get("updateKeyPass"),
            getCustomProperties().get("updateValuePass"));
            almWorkflowUtil.updateAlmTestRunWithAttachment(tc
            Status, getCustomProperties().get("pdfFilePath"),
            getCustomProperties().get("pdfContentType"));
            break;
        default:
```

```
            almWorkflowUtil.updateAlmTestRun(tcStatus,
            getCustomProperties().get("updateKeyFail"),
            getCustomProperties().get("updateValueFail"));
             break;
    }
    almWorkflowUtil.almLogout();
}

}
```

Don't forget to update the step definition file with the new method signature, as in Listing 18-6.

Listing 18-6. Updated Step Definition Method

```
@Then("close application and update HP QC test run")
public void close_application_and_update_HP_QC_test_run()
throws Exception {
    create_pdf_report_update_hpalm();
    tearDown();
}
```

CRUD Operations in TestautomationApplicationTests

Similarly, in the `TestautomationApplicationTests` class, we can add various CRUD operations as we have defined them in the `AlmWorkflowUtil` class. I have added these so that our code looks like Listing 18-7.

Listing 18-7. TestautomationApplicationTests.java

```
package com.taf.testautomation.apitests;

import com.google.gson.JsonElement;
import com.taf.testautomation.TestAutomationProperties;
import com.taf.testautomation.TestautomationApplication;
import com.taf.testautomation.services.RestServices;
import com.taf.testautomation.testmanagement.alm.application.
AlmWorkflowUtil;
import lombok.extern.slf4j.Slf4j;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit.jupiter.
SpringExtension;

@Slf4j
@ExtendWith(SpringExtension.class)
@SpringBootTest(classes = {TestautomationApplication.class})
class TestautomationApplicationTests {
    AlmWorkflowUtil almWorkflowUtil = new AlmWorkflowUtil();

    @Autowired
    TestAutomationProperties testAutomationProperties;

    @Test
    public void testGetServiceCall() throws Exception {
        JsonElement response = RestServices.getJson(testAutomat
        ionProperties.getUrl(), "", "", "sample-service.json");
        log.info(String.valueOf(response));
```

```
if (response.isJsonNull()) {
    Assertions.fail();
}

/**
 * Log in to HP ALM with username and password
 */
almWorkflowUtil.almLogin(testAutomationProperti
es.getAlmUsername(), testAutomationProperties.
getAlmPassword());

/**
 * Get the defect with id x.
 */
almWorkflowUtil.getAlmDefect(testAutomationProperties
.getDefectID());

/**
 * Get the testrun with id y.
 */
almWorkflowUtil.getAlmTestRun(testAutomationProperties
.getTestRunID());

/**
 * Create a new testrun in testlab from existing testset
 */
 almWorkflowUtil.createAlmTestRun();

/**
 * Update the testrun in testlab
 */
almWorkflowUtil.updateAlmTestRun(testAutomation
Properties.getUpdateURL(), testAutomationProperties
.getUpdateKeyPass(), testAutomationProperties
.getUpdateValuePass());
```

```
        almWorkflowUtil.updateAlmTestRunWithAttachme
        nt(testAutomationProperties.getUpdateURL(),
        testAutomationProperties.getPdfFilePath(),
        testAutomationProperties.getPdfContentType());

        /**
         * Delete a testrun in testlab in delete URL
         */
        almWorkflowUtil.deleteAlmTestRun();

        /**
         * Log out of HP ALM
         */
        almWorkflowUtil.almLogout();
    }
}
```

Summary

In this chapter, we learned how to integrate a framework with HP ALM. If you are using HP ALM, I hope you find this chapter very useful. In the next chapter, we will discuss another very interesting topic, localization testing. This will show you how to test your app in multiple supported languages.

CHAPTER 19

Advanced Topic 3: Adding Localization Testing Capabilities

In the previous chapter, we integrated our framework with HP ALM in order to carry out normal CRUD operations. In this chapter, we will discuss how to test localized texts for a multilingual app. You can extend this to also verify non-text elements using reusable methods in the `MobileBaseActionScreen` class (Listing 6-4). This is an advanced value-add chapter and can be skipped if your app is not multilingual. I will first discuss the test approach.

Deciding on Approach Based on Requirements

A multilingual app may support various languages. In such cases, we need to verify various screen elements in all supported languages. This is what we call localization testing. In this book, we will assume there are multiple data files (also called translation files) like the ones (in XML and text) we created to store test data in various languages. So, our `src/test/resources` folder should look like Figure 19-1.

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_19

Figure 19-1. *Data files for localization*

In this case, the `data.txt` or `data.xml` file contains the screen text in various languages. The datatype keys should be replaced with some meaningful key for the particular text. Since we are verifying copyright text and app description strings in this book, let's assume our `data.txt` and `data.xml` files for en-US are as shown in Listing 19-1 and Listing 19-2, respectively.

Listing 19-1. Data.txt

```
/* Localization text for Test Engineer to Architect */
/* Text */
"copyrighttext" = "Data1";
"appdescstrng1" = "Data2";
"appdescstrng2" = "Data3";
```

Listing 19-2. Data.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Localization text for Test Engineer to Architect -->

<resources>
   <!-- Text -->
   <string name="copyrighttext">Data1</string>
   <string name="appdescstrng1">Data2</string>
   <string name="appdescstrng2">Data3</string>
</resources>
```

Localization Testing in Android

In the `FileUtil` class we have already implemented methods to parse lines in a file. We will overload the methods `getCustomValues` and `getInputFilePath` to introduce a new parameter, `language`. The `FileUtil` class is now as shown in Listing 19-3.

Listing 19-3. FileUtil.java

```
package com.taf.testautomation.utilities.fileutil;

import lombok.extern.slf4j.Slf4j;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import java.io.*;
import java.nio.file.Files;
import java.nio.file.Paths;
```

```
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.stream.Collectors;
import java.util.stream.Stream;

import static com.taf.testautomation.utilities.excelutil.Excel
Util.getCustomProperties;


@Slf4j
public class FileUtil {

    private LinkedHashMap<String, String> customSettings =
    new LinkedHashMap<>();
    private static final String INPUT_XML_BASE_PATH =
    "src/test/resources/testdata/xml/";
    private static final String INPUT_TXT_BASE_PATH =
    "src/test/resources/testdata/txt/";
    private static final String XML_FILENAME = "data.xml";
    private static final String DATA_FILENAME = "data.txt";
    private static FileOutputStream fos = null;

    public FileUtil() {
    }

    public Map<String, String> getCustomSettings() {
        String filePath = getInputFilePath();
        try {
            parseXML(filePath, customSettings);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return customSettings;
    }
```

```
public String getCustomValues(String key) {
    String filePath = getInputFilePath();
    String value = "";
    try {
        value = parseFileLines(filePath, key);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return value;
}

public String getCustomValues(String language,
String key) {
    String filePath = getInputFilePath(language);
    String value = "";
    try {
        value = parseFileLines(filePath, key);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return value;
}

private String getInputFilePath() {
    if (getCustomProperties().get("isAndroid")
    .equals("true")) {
        return INPUT_XML_BASE_PATH + XML_FILENAME;
    } else {
        return INPUT_TXT_BASE_PATH + DATA_FILENAME;
    }
}
```

```
    private String getInputFilePath(String language) {
        if (getCustomProperties().get("isAndroid")
        .equals("true")) {
            return INPUT_XML_BASE_PATH + language + "/" +
            XML_FILENAME;
        } else {
            return INPUT_TXT_BASE_PATH + language + "/" + DATA_
            FILENAME;
        }
    }

    private void parseXML(String filePath, LinkedHashMap
    <String, String> customProperties) throws Exception {
        File fXmlFile = new File(filePath);
        DocumentBuilderFactory dbFactory = DocumentBuilder
        Factory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocument
        Builder();
        Document doc = dBuilder.parse(fXmlFile);
        doc.getDocumentElement().normalize();
        log.info("Root element :" + doc.getDocumentElement()
        .getNodeName());
        NodeList nList = doc.getElementsByTagName("string");
        for (int i = 0; i < nList.getLength(); i++) {
            Node nNode = nList.item(i);
            Element eElement = (Element) nNode;
            String key = eElement.getAttribute("name");
            String value = nNode.getTextContent();
            customProperties.put(key, value);
        }
    }
```

```
private String parseFileLines(String filePath, String key) {
    String strLine = "";
    try {
        Stream<String> lines = Files.lines(Paths.get
        (filePath));
        strLine = lines.filter(s -> s.contains(key)).
        collect(Collectors.toList()).get(0);
    } catch (Exception e) {
        e.printStackTrace();
    }
    if (getCustomProperties().get("isAndroid").equals
    ("true")) {
        return strLine.substring(strLine.indexOf(">") + 1,
        strLine.lastIndexOf("<"));
    } else {
        return strLine.substring(strLine.indexOf("=") + 3,
        strLine.indexOf(";") - 1);
    }
}

public static void fileCopy(String inputFile,
String outputFile) {
    String cleanLine = "";

    try {
        fos = new FileOutputStream(outputFile);
        FileInputStream fileInputStream = new FileInput
        Stream(inputFile);
        BufferedReader br = new BufferedReader(new Input
        StreamReader(fileInputStream));
        String strLine = null;
        int linesCounter = 0;
```

```
            while ((strLine = br.readLine()) != null) {// read
            every line in the file
                cleanLine += strLine + "\n";
            }
            char[] stringToCharArray = cleanLine.toCharArray();
            for (char ch : stringToCharArray)
                fos.write(ch);
            fos.close();
        } catch (IOException e) {
        }
    }
}
```

We will now utilize these methods to perform localization testing for the target strings. Let's create two new methods in the `AboutAppTestSuite` class, namely `testScenario6()` and `testScenario7()`, one to verify copyright text and the other to verify app description strings. The new test methods are shown in Listing 19-4.

Listing 19-4. Methods to Verify Texts in Multiple Locales

```
@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the copyright text is displayed")
@Test
@Order(6)
@Smoke
@Regression
@SIT
@AT
public void testScenario6() throws Exception {
```

```
String tcName = new Object() {
}.getClass().getEnclosingMethod().getName();
log("Test Name" + tcName);

String copyrightTxt = "";
String languageCodes = getCustomProperties().get
("deviceLanguage");
List<String> list =
        Stream.of(languageCodes.split("\\s*,\\s*"))
        .collect(Collectors.toList());
for (String str : list) {
    try {
        log("Language code is: " + str.substring
        (0, 2) + "\n" + "Country code is: " + str.
        substring(3));
        if (getCustomProperties().get("isAndroid")
        .equals("true")) {
            appiumUtil.changeLanguageADB(str.substring
            (0, 2), str.substring(3));
        } else {
            session.getAppiumDriver().quit();
            getCustomProperties().put
            ("localization", "yes");
            getCustomProperties().put
            ("appLanguage", str);
            session = startDefaultSession();
        aboutAppScreen = new    ScreenNavigation
        (session).getAboutAppScreenFromAccountCreation
        Screen(getCustomProperties().get("username"),
        getCustomProperties().get("password"));
        }
```

```
            fileUtil = new FileUtil();
            copyrightTxt = fileUtil.getCustomValues
            (str, "copyrighttext");
            SoftAssertions.assertSoftly(
                    softAssertions -> {
                        softAssertions.assertThat(aboutApp
                        Screen.isCopyrightTextDisplayed
                        (fileUtil.getCustomValues(str,
                        "copyrighttext"))).as("The Copyright is
                        correctly displayed").isTrue();
                    }
            );
        } finally {
            testStatus = aboutAppScreen.isCopyrightTextDisplayed
            (copyrightTxt) ? "Passed" : "Failed";
            updateTCPassCount();
            try {
                aboutAppScreen.takeScreenShot("test-result/
                screenshots/" + tcName + "-" + str +
                copyrightTxt + "--" + testStatus + ".png");
            } catch (Exception e) {
                e.printStackTrace();
            }
            imageList.add("test-result/screenshots/" + tcName
            + "-" + str + copyrightTxt + "--" + testStatus +
            ".png");
        }
    }
}

@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
```

```
@DisplayName("xxxx")
@Description("xxxx: Verify that the App description texts are
displayed")
@Test
@Order(7)
@Smoke
@Regression
@SIT
@AT
public void testScenario7() throws Exception {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    String appDescTxt1 = "", appDescTxt2 = "";
    String languageCodes = getCustomProperties().
    get("deviceLanguage");
    List<String> list =
            Stream.of(languageCodes.split("\\s*,\\s*"))
            .collect(Collectors.toList());
    for (String str : list) {
        try {
            log("Language code is: " + str.substring(0, 2) +
            "\n" + "Country code is: " + str.substring(3));
            if (getCustomProperties().get("isAndroid")
            .equals("true")) {
                appiumUtil.changeLanguageADB
                (str.substring(0, 2), str.substring(3));
            } else {
                session.getAppiumDriver().quit();
                getCustomProperties().put
                ("localization", "yes");
```

```
                getCustomProperties().put
                ("appLanguage", str);
                session = startDefaultSession();
                aboutAppScreen = new    ScreenNavigation
                (session).getAboutAppScreenFromAccountCreatio
                nScreen(getCustomProperties().get
                ("username"), getCustomProperties().get
                ("password"));
            }
            fileUtil = new FileUtil();
            appDescTxt1 = fileUtil.getCustomValues
            (str, "appdescstrng1");
            appDescTxt2 = fileUtil.getCustomValues
            (str, "appdescstrng2");
            SoftAssertions.assertSoftly(
                    softAssertions -> {
                        softAssertions.assertThat
                        (aboutAppScreen.isAppDescr
                        iptionDisplayed(fileUtil.
                        getCustomValues(str, "appdescstrng1"),
                        fileUtil.getCustomValues(str,
                        "appdescstrng2"))).as
                        ("The App description is correctly
                        displayed").isTrue();
                    }
            );
        } finally {
            testStatus = aboutAppScreen.isAppDescriptionDisplay
            ed(appDescTxt1, appDescTxt2) ? "Passed" : "Failed";
            updateTCPassCount();
            try {
```

```
                aboutAppScreen.takeScreenShot("test-result/
                screenshots/" + tcName + "-" + str +
                appDescTxt1 + "##" + appDescTxt2 + "--" +
                testStatus + ".png");
            } catch (Exception e) {
                e.printStackTrace();
            }
            imageList.add("test-result/screenshots/" + tcName +
            "-" + str + appDescTxt1 + "##" + appDescTxt2 + "--"
            + testStatus + ".png");
        }
    }
}
```

In `testScenario6()` and `testScenario7()`, we first read the `deviceLanguage` property key, retrieving individual `language-Locale` entries and then storing those in a `List` of `String`. Then we iterate over this list, and for each entry we call the `changeLanguageADB(String language, String country)` method that we implemented in the `AppiumUtil` class. Then, we use the newly overloaded `getCustomValues` method in the `FileUtil` class to fetch the value for the copyright and app description key. And finally, we call the page object method to verify the presence of target strings.

You might note that in the `assertThat` method I have not used the local variable created for the copyright and app description texts. That's because local variables are not allowed in lambda expressions.

Now, let's discuss localization testing in iOS.

Localization Testing in iOS

Localization testing in iOS is similar to that in Android except that we use the `language` and `locale desiredCapabilities` to change the language at the app level only. In Android, we change the language at the device level. We do this in iOS by changing the values of the property parameters `localization` and `appLanguage` dynamically in the map where we store them. Then, we need to recreate the driver to launch the app in the desired language. Once we launch, we need to navigate back to our landing screen, which is `AboutAppScreen`. Please keep in mind that we need to write generic locators that would work in multiple languages for navigation. One example is shown in Listing 19-5.

Listing 19-5. Generic Locators for Multiple Languages

```
@AndroidFindBy(xpath = "//android.widget.LinearLayout/android.
widget.FrameLayout/android.view.ViewGroup")
@iOSXCUITFindBy((xpath = "//XCUIElementTypeCollectionView/
XCUIElementTypeCell/XCUIElementTypeOther")
private MobileElement someMobileElement;
```

With the addition of these methods to the test suite, we need to change the BDD artifacts. These are `AboutApp.feature` and `AboutAppStepDefinitions.java`, as shown in Listing 19-6 and Listing 19-7, respectively.

Listing 19-6. AboutApp.feature

```
@AboutApp
Feature: AboutApp
  Req number: xxxx

  Background:
    Given application is installed and launched
```

```
@AboutApp @Regression
Scenario: client wants to verify about app screen elements
  When user opens the application
  Then verify create account screen is displayed
  When user fills in details
  And clicks on Create Account button
  Then verified user is taken to the app login screen
  When user logs in with email and password
  And clicks on Sign In button
  Then verified user is taken to the about app screen
  And verified user sees the following in-app screen
    | Screen_Title    |
    | App_Logo        |
    | App_Name        |
    | App_Version     |
    | App_Images      |
    | Copyright_Txt   |
    | App_Description |
  Then close application and update HP QC test run
```

Listing 19-7. AboutAppStepDefinitions.java

```
package com.taf.testautomation.cucumber.aboutapp;

import io.cucumber.datatable.DataTable;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;

import java.util.List;
```

```
public class AboutAppStepDefinitions extends
SpringIntegration {

    @Given("application is installed and launched")
    public void application_is_installed_and_launched()
    throws Exception {
        setUp();
    }

    @When("user opens the application")
    public void user_opens_the_application() {
        navigateToScreen();
    }

    @Then("verify create account screen is displayed")
    public void verify_create_account_screen_is_displayed() {
        log("leaving the implementation to reader");
    }

    @When("user fills in details")
    public void user_fills_in_details() {
        log("this step is covered in navigateToScreen()");
    }

    @And("clicks on Create Account button")
    public void clicks_on_create_account_button() {
        log("this step is covered in navigateToScreen()");
    }

    @Then("verified user is taken to the app login screen")
    public void verify_user_is_taken_to_the_app_login_screen() {
        log("leaving the implementation to reader");
    }
```

```
@When("user logs in with email and password")
public void user_logs_in_with_email_and_password() {
    log("this step is covered in navigateToScreen()");
}

@And("clicks on Sign In button")
public void clicks_on_sign_in_button() {
    log("this step is covered in navigateToScreen()");
}

@Then("verified user is taken to the about app screen")
public void verify_user_is_taken_to_the_about_app_screen() {
    log("leaving the implementation to reader");
}

@And("verified user sees the following in app screen")
public void verify_user_sees_the_following_in_app_
screen(DataTable dt) {
    List<String> list = dt.asList(String.class);
    dataTable = new String[list.size()][1];
    for (int i = 0; i < list.size(); i++) {
        dataTable[i][0] = list.get(i);
    }
    for (String str : list) {
        switch (str) {
            case "Screen_Title":
                testScenario1();
                break;
            case "App_Logo":
                testScenario2();
                break;
            case "App_Name":
                testScenario3();
```

```
                    break;
                case "App_Version":
                    testScenario4();
                    break;
                case "App_Images":
                    testScenario5();
                    break;
                case "Copyright_Txt":
                    try {
                        testScenario6();
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    break;
                default:
                    try {
                        testScenario7();
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    break;
            }
        }
    }

    @Then("close application and update HP QC test run")
    public void close_application_and_update_HP_QC_test_run()
    throws Exception {
        create_pdf_report_update_hpalm();
        tearDown();
    }

}
```

Summary

In this chapter, we learned how we can test the text elements of a multilingual app. The presence and alignment of non-text elements can be verified with our standard methods in the `MobileBaseActionScreen` class (Listing 6-4). In the next chapter, we will discuss a recent yet highly popular topic—parallel test execution.

CHAPTER 20

Advanced Topic 4: Implementing Parallel Test Execution

In the previous chapter, we added localization testing capability to our framework, which allows us to test a multilingual app. In this chapter, we will discuss how to spawn and manage multiple driver instances. This will be useful when running tests in a device cloud for parallel execution. Let's see how to manage multiple sessions in the next section.

Managing Multiple Sessions

Parallel execution requires the triggering of the test suite in multiple devices, usually hosted in device clouds by providers like Experitest, BrowserStack, HeadSpin, Kobiton, and so on. This is highly popular in today's automation projects. In Chapter 3, while configuring Gradle, we added properties for parallel execution to `build.gradle`. In the parallel execution setup, the agent process triggers the `setUp()` method in each device that is discovered by device queries. Please refer to `uitests.properties` for `SeeTest`-related queries. This means creating and managing multiple session objects in the framework.

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0_20

So far, we have created "stateless" session objects in the sense that we do not remember the object in our code after it has been created. But for parallel execution we need to identify a particular session object. So, let's assign each session object a name once it has been created and hold the name–object pair in a map of `<String, Object>` type. Given a set of available session names in a list, we need to assign a session name that is available and not already assigned to an existing object. Similarly, when closing a session, we need to quit the relevant driver instance. Let's implement these methods in a new class called `SessionManagement`, as shown in Listing 20-1.

Listing 20-1. SessionManagement.java

```
package com.taf.testautomation;

import lombok.extern.slf4j.Slf4j;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.function.Consumer;

@Slf4j
public class SessionManagement {

    private HashMap<String, Session> sessionList;
    private String currentSessionName;

    public SessionManagement() {
        this.sessionList = new HashMap<String, Session>();
    }

    public Session startSessionWithName(String name) {
        synchronized (this.sessionList) {
            if (this.sessionList.containsKey(name)) {
```

```
                throw new IllegalArgumentException("Session
                already started, please close Session before
                starting a new Session with same name:" + name);
            }
        }
        Session session = new Session();
        try {
            session.startSession();
            synchronized (this.sessionList) {
                this.sessionList.put(name, session);
            }
            this.setCurrentSessionName(name);
        } catch (Exception e) {
            log.info("Error starting session:" + "\n" + e);
            throw new RuntimeException(e);
        }
        return session;
    }

    public Session startFirstAvailableSession(List<String>
    sessionNames) {
        String sessionName = sessionNames.stream().filter
        (e -> !new ArrayList<String>(this.sessionList.
        keySet()).contains(e)).findFirst().orElse(null);
        Session session = new Session();
        try {
            session.startSession();
            synchronized (this.sessionList) {
                this.sessionList.put(sessionName, session);
            }
            this.setCurrentSessionName(sessionName);
        } catch (Exception e) {
```

```
            log.info("Error starting session:"+ "\n" + e);
            throw new RuntimeException(e);
        }
        log.info("Available Session names are " + new
        ArrayList<String>(this.sessionList.keySet()));
        return session;
    }

    public Session getCurrentSession() {
        String currentSessionName = this.getCurrentSessionName();
        return this.getSession(currentSessionName);
    }

    public Session getSession(String sessionName) {
        synchronized (this.sessionList) {
            return this.sessionList.get(this.currentSessionName);
        }
    }

    public void closeSession(String sessionName) {
        Session session;
        synchronized (this.sessionList) {
            session = this.sessionList.get(sessionName);
        }
        this.closeSession(sessionName, session);
    }

    private void closeSession(String name, Session session) {
        try {
            session.closeSession();
            synchronized (this.sessionList) {
                this.sessionList.remove(name);
            }
```

```
            } catch (Exception e) {
                log.info("Error closing session", e);
            }
        }

        public void closeAllSessions() {
            List<String> sessions;
            synchronized (this.sessionList) {
                sessions = new ArrayList<String>(this.sessionList.
                keySet());
            }
            sessions.parallelStream().forEach((Consumer<? super
            Object>) e -> closeSession((String) e));
        }

        public HashMap<String, Session> getSessionList() {
            return this.sessionList;
        }

        public String getCurrentSessionName() {
            return this.currentSessionName;
        }

        public void setSessionList(HashMap<String, Session>
        sessionList) {
            this.sessionList = sessionList;
        }

        public void setCurrentSessionName(String currentSessionName) {
            this.currentSessionName = currentSessionName;
        }
    }
```

The first method creates a session object with a given name. The second method, `startFirstAvailableSession(List<String> sessionNames)`, finds the first available session name from the `sessionNames` list by checking the list created from the session name keys. Then it assigns this name to the session object created. The overloaded `closeSession` methods work together to close a session with a given session name. The `closeAllSessions()` method closes all the active sessions. The getter and setter methods are self-explanatory. Now it is time to update the `BaseTest` class.

Updating BaseTest Class

We call the `setup()` and `teardown()` methods from `BaseTest`. The `BaseTest` class must be updated accordingly, as shown in Listing 20-2.

Listing 20-2. BaseTest.java

```
package com.taf.testautomation;

import lombok.Getter;
import lombok.Setter;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Arrays;
import java.util.HashMap;
import java.util.List;

@Getter
@Setter
@SuppressWarnings("rawtypes")
```

```
public class BaseTest {

    protected Session session = new Session();
    protected String sessionName;
    protected SessionManagement sessionManagement = new
    SessionManagement();
    protected HashMap<String, String> customProperties =
    session.getCustomProperties();
    private Logger logger = LoggerFactory.getLogger(this.
    getClass());
    protected static String[][] dataTable;
    protected static List<String> sessionNames =
    Arrays.asList("Session1", "Session2", "Session3",
    "Session4", "Session5");

    @BeforeAll
    public void setUp() throws Exception {
        log("Initializing Session");
        session = startSession(sessionNames);
        sessionName = sessionManagement.getCurrentSessionName();
        log("Session created");
    }

    @AfterAll
    public void tearDown() throws Exception {
        log("Destroying Session");
        closeAllSessions();
        log("Session destroyed");
    }

    public void log(String message) {
        getLogger().info(message);
    }
```

```
    public void logError(String message) {
        getLogger().error(message);
    }

    public Session startSession(String sessionName) {
        try {
            sessionManagement.startSessionWithName(sessionName);
        } catch (Exception e) {
            logError("Error starting Session, trying again" +
            e.getMessage());
            sessionManagement.startSessionWithName(sessionName);
        }
        if (sessionManagement.getCurrentSession().getAppium
        Driver() != null) {
            return sessionManagement.getCurrentSession();
        } else throw new IllegalArgumentException("Error
        starting Session:" + sessionName);
    }

    public Session startSession(List<String> sessionNames) {
        try {
            sessionManagement.startFirstAvailableSession
            (sessionNames);
        } catch (Exception e) {
            logError("Error starting Session, trying again"
            + e.getMessage());
            sessionManagement.startFirstAvailableSession
            (sessionNames);
        }
```

```
        if (sessionManagement.getCurrentSession().
        getAppiumDriver() != null) {
            return sessionManagement.getCurrentSession();
        } else throw new IllegalArgumentException("Error
        starting Session:");
    }

    public void closeSession(String sessionName) {
        try {
            sessionManagement.closeSession(sessionName);
        } catch (Exception e) {
            throw new IllegalArgumentException("Error closing
            Session:" + sessionName);
        }
    }

    public void closeAllSessions() {
        try {
            sessionManagement.closeAllSessions();
        } catch (Exception e) {
            throw new IllegalArgumentException("Error closing
            Sessions:" + e.getMessage());
        }
    }
}
```

We declare and initialize the `sessionNames` list in `BaseTest` and pass this as a parameter to the corresponding `startSession` method. We do not invoke `Session` methods directly from `BaseTest`. Instead, we do so through the `SessionManagement` class, and we update the value of `sessionName` each time we create an object.

Now we need to update the test suite.

Updating Test Suites and Step Definitions

In our test suite file, we need to update the code where we quit and recreate the driver. This also needs to be done in the step definition, if applicable (in our case, it is not). In our `AboutAppTestSuite` example, `testScenario3()`, `testScenario4()`, `testScenario6()`, and `testScenario7()` need to be updated, as shown in Listing 20-3.

Listing 20-3. Updated Methods in AboutAppTestSuite.java

```
@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the App Name is displayed")
@Test
@Order(3)
@Regression
public void testScenario3() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    if (session.getAppiumDriver().removeApp
    (getCustomProperties().get("appPackage"))) {
        try {
            closeSession(sessionName);
            session.setPrevBuild("yes");
            session = startSession(sessionNames);
            sessionName = sessionManagement.getCurrent
            SessionName();
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }
    try {
        SoftAssertions.assertSoftly(
                softAssertions -> {
                    softAssertions.assertThat
                    (aboutAppScreen.isAppNameDisplayed()).
                    as("The App Name is displayed").isTrue();
                }
        );
    } finally {
        testStatus = aboutAppScreen.isAppNameDisplayed() ?
        "Passed" : "Failed";
        updateTCPassCount();
        try {
            aboutAppScreen.takeScreenShot("test-result/
            screenshots/" + tcName + "--" + testStatus +
            ".png");
        } catch (Exception e) {
            e.printStackTrace();
        }
        imageList.add("test-result/screenshots/" + tcName
        + "--" + testStatus + ".png");
    }
}

@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the App Version is displayed")
@Test
@Order(4)
@SIT
```

```
public void testScenario4() {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    String app = getCustomProperties().get("app2");
    closeSession(sessionName);
    getCustomProperties().put("appPackage", "xxxx");
    getCustomProperties().put("app", app);
    try {
        session = startSession(sessionNames);
        sessionName = sessionManagement.
        getCurrentSessionName();
    } catch (Exception e) {
        e.printStackTrace();
    }
    try {
        SoftAssertions.assertSoftly(
                softAssertions -> {
                    softAssertions.assertThat
                    (aboutAppScreen.isAppVersionDisplayed
                    ("xxxx")).as("The App Version is
                    displayed").isTrue();
                }
        );
    } finally {
        testStatus = aboutAppScreen.isAppVersionDisplayed
        ("xxxx") ? "Passed" : "Failed";
        updateTCPassCount();
        try {
            aboutAppScreen.takeScreenShot("test-result/
            screenshots/" + tcName + "--" + testStatus + ".png");
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
        imageList.add("test-result/screenshots/" + tcName +
        "--" + testStatus + ".png");
    }
}

@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the copyright text is displayed")
@Test
@Order(6)
@Smoke
@Regression
@SIT
@AT
public void testScenario6() throws Exception {
    String tcName = new Object() {
    }.getClass().getEnclosingMethod().getName();
    log("Test Name" + tcName);

    String copyrightTxt = "";
    String languageCodes = getCustomProperties().get
    ("deviceLanguage");
    List<String> list =
            Stream.of(languageCodes.split("\\s*,\\s*"))
            .collect(Collectors.toList());
    for (String str : list) {
        try {
```

```
log("Language code is: " + str.substring
(0, 2) + "\n" + "Country code is: " + str.
substring(3));
if (getCustomProperties().get("isAndroid").
equals("true")) {
    appiumUtil.changeLanguageADB(str.substring
    (0, 2), str.substring(3));
} else {
    closeSession(sessionName);
    getCustomProperties().put("localization", "yes");
    getCustomProperties().put("appLanguage", str);
    session = startSession(sessionNames);
    sessionName = sessionManagement.getCurrent
    SessionName();
    aboutAppScreen = new    ScreenNavigation
    (session).getAboutAppScreenFromAccountCreation
    Screen(getCustomProperties().get("username"),
    getCustomProperties().get("password"));
}
fileUtil = new FileUtil();
copyrightTxt = fileUtil.getCustomValues
(str, "copyrighttext");
SoftAssertions.assertSoftly(
        softAssertions -> {
            softAssertions.assertThat
            (aboutAppScreen.isCopyrightTextDisp
            layed(fileUtil.getCustomValues(str,
            "copyrighttext"))).as("The Copyright
            is correctly displayed").isTrue();
```

```
                    }
            );
        } finally {
            testStatus = aboutAppScreen.isCopyrightTextDisplayed
            (copyrightTxt) ? "Passed" : "Failed";
            updateTCPassCount();
            try {
                aboutAppScreen.takeScreenShot("test-result/
                screenshots/" + tcName + "-" + str +
                copyrightTxt + "--" + testStatus + ".png");
            } catch (Exception e) {
                e.printStackTrace();
            }
            imageList.add("test-result/screenshots/" + tcName
            + "-" + str + copyrightTxt + "--" + testStatus +
            ".png");
        }
    }
}

@Severity(SeverityLevel.CRITICAL)
@Issue("xxxx")
@DisplayName("xxxx")
@Description("xxxx: Verify that the App description texts are
displayed")
@Test
@Order(7)
@Smoke
@Regression
@SIT
@AT
public void testScenario7() throws Exception {
```

```
String tcName = new Object() {
}.getClass().getEnclosingMethod().getName();
log("Test Name" + tcName);

String appDescTxt1 = "", appDescTxt2 = "";
String languageCodes = getCustomProperties().get
("deviceLanguage");
List<String> list =
        Stream.of(languageCodes.split("\\s*,\\s*"))
        .collect(Collectors.toList());
for (String str : list) {
    try {
        log("Language code is: " + str.substring
        (0, 2) + "\n" + "Country code is: " + str.
        substring(3));
        if (getCustomProperties().get("isAndroid").
        equals("true")) {
            appiumUtil.changeLanguageADB
            (str.substring(0, 2), str.substring(3));
        } else {
            closeSession(sessionName);
            getCustomProperties().put
            ("localization", "yes");
            getCustomProperties().put
            ("appLanguage", str);
            session = startSession(sessionNames);
            sessionName = sessionManagement.
            getCurrentSessionName();
            aboutAppScreen = new    ScreenNavigation
            (session).getAboutAppScreenFromAccountCreation
            Screen(getCustomProperties().get("username"),
            getCustomProperties().get("password"));
```

```
    }
    fileUtil = new FileUtil();
    appDescTxt1 = fileUtil.getCustomValues
    (str, "appdescstrng1");
    appDescTxt2 = fileUtil.getCustomValues
    (str, "appdescstrng2");
    SoftAssertions.assertSoftly(
            softAssertions -> {
                softAssertions.assertThat
                (aboutAppScreen.isAppDescriptionDisp
                layed(fileUtil.getCustomValues(str,
                "appdescstrng1"), fileUtil.get
                CustomValues(str, "appdescstrng2")))
                .as("The App description is correctly
                displayed").isTrue();
            }
    );
} finally {
    testStatus = aboutAppScreen.isAppDescriptionDisplayed
    (appDescTxt1, appDescTxt2) ? "Passed" : "Failed";
    updateTCPassCount();
    try {
        aboutAppScreen.takeScreenShot("test-result/
        screenshots/" + tcName + "-" + str +
        appDescTxt1 + "##" + appDescTxt2 + "--" +
        testStatus + ".png");
    } catch (Exception e) {
        e.printStackTrace();
    }
```

```
            imageList.add("test-result/screenshots/" + tcName +
            "-" + str + appDescTxt1 + "##" + appDescTxt2 + "--"
            + testStatus + ".png");
        }
    }
}
```

Summary

In this chapter, we learned how to create and handle multiple session objects for parallel test execution.

We started the book with a framework overview and discussed how to set up the core components in the first six chapters. We built the page object model and test layer from Chapters 7 to 16. Subsequently, we covered a few advanced topics, starting in Chapter 17. I hope you are now fully confident in taking on the role of automation architect! However, we are not done yet. In the Appendix, we will revisit the automation setup and a few more utilities.

APPENDIX A

Other Utilities

Throughout this book, my focus has been on preparing the reader to think and act like a hands-on architect. I hope you have enjoyed reading every bit of it. In this appendix, we will discuss a few other utilities that you may find useful. Let's start with OCR Util.

OCR Util

We will use the Tesseract library to read text from an image. While Python and C++ have an integrated Tesseract library, in Java we use the tess4J dependency, which we added via build.gradle. In addition, we need to install the Tesseract package. On a Mac, you can use brew to install it: `brew install tesseract`. Once installed, create a class `OcrUtil` as shown in Listing A-1 inside subfolder `ocrutil` within the `utilities` folder.

Listing A-1. OcrUtil.java

```
package com.taf.testautomation.utilities.ocrutil;

import net.sourceforge.tess4j.Tesseract;
import net.sourceforge.tess4j.TesseractException;

import java.io.File;

public class OcrUtil {
```


K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0

```
    private static String result = "";
    private static Tesseract tesseract = new Tesseract();

    static {
        System.setProperty("jna.library.path", "/usr/local/
        Cellar/tesseract/4.1.1/lib/");
        tesseract.setDatapath("./tessdata");
    }

    public static String readImage() {
        try {
            result = tesseract.doOCR(new File
            ("./screenshot.png"));
        } catch (TesseractException e) {
            e.printStackTrace();
        }
        return result;
    }
}
```

The `doOCR(File file)` method returns the text from screenshot.png in the project's `root` folder. You can write a different method or overload this method to read images from specific folders.

Image Comparison Util

In this section, we will create a class `ImageComparison` inside the `imagecomparison` subfolder of the `utilities` folder. In this class we will compare two images pixel by pixel. The reference image is stored in a fixed location, the `src/test/resources/testimages` folder, whereas the image to be validated is in the project `root` or another folder. Image comparison is useful if you are doing screenshot verification in a page object class. The code for this class is shown in Listing A-2.

Listing A-2. ImageComparison.java

```
package com.taf.testautomation.utilities.imagecomparison;

import lombok.extern.slf4j.Slf4j;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

@Slf4j
public class ImageComparison {

    private static BufferedImage img1 = null;
    private static BufferedImage img2 = null;
    private static double percentage = 0.0;

    public static double compareImages(String image1, String
    image2) {

        String image1Path = "./" + image1;
        String image2Path = "./src/test/resources/testimages/"
        + image2;

        try {
            File file1 = new File(image1Path);
            File file2 = new File(image2Path);
            img1 = ImageIO.read(file1);
            img2 = ImageIO.read(file2);
        } catch (IOException e) {
            log.error(e.getMessage());
        }

        int width1 = img1.getWidth();
        int width2 = img2.getWidth();
```

```
int height1 = img1.getHeight();
int height2 = img2.getHeight();
if ((width1 != width2) || (height1 != height2))
    log.error("Error: Image dimensions mismatch");
else {
    long difference = 0;
    for (int y = 0; y < height1; y++) {
        for (int x = 0; x < width1; x++) {
            int rgbA = img1.getRGB(x, y);
            int rgbB = img2.getRGB(x, y);
            int redA = (rgbA >> 16) & 0xff;
            int greenA = (rgbA >> 8) & 0xff;
            int blueA = (rgbA) & 0xff;
            int redB = (rgbB >> 16) & 0xff;
            int greenB = (rgbB >> 8) & 0xff;
            int blueB = (rgbB) & 0xff;
            difference += Math.abs(redA - redB);
            difference += Math.abs(greenA - greenB);
            difference += Math.abs(blueA - blueB);
        }
    }

    double total_pixels = width1 * height1 * 3;

    double avg_different_pixels = difference /
            total_pixels;

    percentage = (avg_different_pixels /
            255) * 100;
}
log.info("The difference in percentage for image
comparison" + percentage);
```

```
        return percentage;
    }
}
```

As you can see, the method `compareImages` returns the percentage difference between the two images. So, to return a Boolean value, the user can check for this percentage difference to be less than the maximum value allowed.

Email Util

Now we will write an `EmailUtil` class with a few methods to send text and HTML emails, send emails with attachments, and send extent reports generated from BDD execution to designated email IDs. For this purpose, I have reused the examples from https://mkyong.com/java/java-how-to-send-email/ and modified them to suit our purpose. Listing A-3 shows the code for the `EmailUtil` class.

Listing A-3. EmailUtil.java

```
package com.taf.testautomation.utilities.emailutil;

import com.sun.mail.smtp.SMTPTransport;
import lombok.extern.slf4j.Slf4j;

import javax.activation.DataHandler;
import javax.activation.DataSource;
import javax.activation.FileDataSource;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Multipart;
import javax.mail.Session;
import javax.mail.internet.InternetAddress;
```

```
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Date;
import java.util.Properties;

import static com.taf.testautomation.utilities.excelutil.ExcelU
til.getCustomProperties;


@Slf4j
public class EmailUtil {

    private static final String SMTP_SERVER = "smtp-mail.
    outlook.com";
    private static final String USERNAME = "xxxx@hotmail.com";
    private static final String PASSWORD = "xxxx";

    private static final String EMAIL_FROM =
    "xxxx@hotmail.com";
    private static final String EMAIL_TO =
    "xxxx@organization.com";
    private static final String EMAIL_TO_CC = "xxxx@gmail.com";

    private static final String EMAIL_SUBJECT = "Test Execution
    Notification: ";

    public static void sendTextEmail(String content, String
    subject) {

        Properties prop = System.getProperties();
```

```
prop.put("mail.smtp.host", SMTP_SERVER);
prop.put("mail.smtp.auth", "true");
prop.put("mail.smtp.port", "587");
prop.put("mail.smtp.starttls.enable", "true");

Session session = Session.getInstance(prop, null);
Message msg = new MimeMessage(session);
String emailSubject = EMAIL_SUBJECT + subject;

try {
    // from
    msg.setFrom(new InternetAddress(EMAIL_FROM));

    // to
    msg.setRecipients(Message.RecipientType.TO,
            InternetAddress.parse(EMAIL_TO, false));

    // cc
    msg.setRecipients(Message.RecipientType.CC,
            InternetAddress.parse(EMAIL_TO_CC, false));

    // subject
    msg.setSubject(emailSubject);

    // content
    msg.setText(content);

    msg.setSentDate(new Date());

    // Get SMTPTransport
    SMTPTransport t = (SMTPTransport) session.
    getTransport("smtp");

    // connect
    t.connect(SMTP_SERVER, USERNAME, PASSWORD);
```

```
            // send
            t.sendMessage(msg, msg.getAllRecipients());

            log.info("Response: " + t.getLastServerResponse());

            t.close();

        } catch (MessagingException e) {
            e.printStackTrace();
        }
    }

    public static void sendHtmlEmail(String content, String
    subject) {

        Properties prop = System.getProperties();

        prop.put("mail.smtp.host", SMTP_SERVER);
        prop.put("mail.smtp.auth", "true");
        prop.put("mail.smtp.port", "587");
        prop.put("mail.smtp.starttls.enable", "true");

        Session session = Session.getInstance(prop, null);
        Message msg = new MimeMessage(session);
        String emailSubject = EMAIL_SUBJECT + subject;

        try {

            msg.setFrom(new InternetAddress(EMAIL_FROM));

            msg.setRecipients(Message.RecipientType.TO,
                    InternetAddress.parse(EMAIL_TO, false));

            msg.setSubject(emailSubject);

            // HTML email
```

```
            msg.setDataHandler(new DataHandler(new
            HTMLDataSource(content)));

            SMTPTransport t = (SMTPTransport) session.
            getTransport("smtp");

            // connect
            t.connect(SMTP_SERVER, USERNAME, PASSWORD);

            // send
            t.sendMessage(msg, msg.getAllRecipients());

            log.info("Response: " + t.getLastServerResponse());

            t.close();

        } catch (MessagingException e) {
            e.printStackTrace();
        }

}

public static void sendEmailAttachment(String content,
String subject) {

    Properties prop = System.getProperties();

    prop.put("mail.smtp.host", SMTP_SERVER);
    prop.put("mail.smtp.auth", "true");
    prop.put("mail.smtp.port", "587");
    prop.put("mail.smtp.starttls.enable", "true");
    String emailSubject = EMAIL_SUBJECT + subject;

    Session session = Session.getInstance(prop, null);
    Message msg = new MimeMessage(session);

    try {
```

```
msg.setFrom(new InternetAddress(EMAIL_FROM));

msg.setRecipients(Message.RecipientType.TO,
        InternetAddress.parse(EMAIL_TO, false));

msg.setSubject(emailSubject);

// text
MimeBodyPart p1 = new MimeBodyPart();
p1.setText(content);

// file
MimeBodyPart p3 = new MimeBodyPart();
FileDataSource fds2 = new FileDataSource(getCustomP
roperties().get("mergedReport"));
p3.setDataHandler(new DataHandler(fds2));
p3.setFileName(fds2.getName());

Multipart mp = new MimeMultipart();
mp.addBodyPart(p1);
mp.addBodyPart(p3);

msg.setContent(mp);

SMTPTransport t = (SMTPTransport) session.
getTransport("smtp");

// connect
t.connect(SMTP_SERVER, USERNAME, PASSWORD);

// send
t.sendMessage(msg, msg.getAllRecipients());

log.info("Response: " + t.getLastServerResponse());

t.close();
```

```
    } catch (MessagingException e) {
        e.printStackTrace();
    }

}

public static void sendExtentReport(String content, String
subject) {

    Properties prop = System.getProperties();

    prop.put("mail.smtp.host", SMTP_SERVER);
    prop.put("mail.smtp.auth", "true");
    prop.put("mail.smtp.port", "587");
    prop.put("mail.smtp.starttls.enable", "true");
    String emailSubject = EMAIL_SUBJECT + subject;

    Session session = Session.getInstance(prop, null);
    Message msg = new MimeMessage(session);

    try {

        msg.setFrom(new InternetAddress(EMAIL_FROM));

        msg.setRecipients(Message.RecipientType.TO,
                InternetAddress.parse(EMAIL_TO, false));

        msg.setSubject(emailSubject);

        // text
        MimeBodyPart p1 = new MimeBodyPart();
        p1.setText(content);

        // file
        MimeBodyPart p2 = new MimeBodyPart();
        FileDataSource fds = new FileDataSource("./test-
        output/HtmlReport/ExtentHtml.html");
```

```
            p2.setDataHandler(new DataHandler(fds));
            p2.setFileName(fds.getName());

            Multipart mp = new MimeMultipart();
            mp.addBodyPart(p1);
            mp.addBodyPart(p2);

            msg.setContent(mp);

            SMTPTransport t = (SMTPTransport) session.
            getTransport("smtp");

            // connect
            t.connect(SMTP_SERVER, USERNAME, PASSWORD);

            // send
            t.sendMessage(msg, msg.getAllRecipients());

            log.info("Response: " + t.getLastServerResponse());

            t.close();

        } catch (MessagingException e) {
            e.printStackTrace();
        }

    }

    static class HTMLDataSource implements DataSource {

        private String html;

        public HTMLDataSource(String htmlString) {
            html = htmlString;
        }

        @Override
```

```
        public InputStream getInputStream() throws
        IOException {
            if (html == null) throw new IOException("html
            message is null!");
            return new ByteArrayInputStream(html.getBytes());
        }

        @Override
        public OutputStream getOutputStream() throws
        IOException {
            throw new IOException("This DataHandler cannot
            write HTML");
        }

        @Override
        public String getContentType() {
            return "text/html";
        }

        @Override
        public String getName() {
            return "HTMLDataSource";
        }
    }
}
```

I will ask you to go through the preceding link for the first three methods. We use `MimeBodyPart` objects from the javax.mail package to store email content. *Mime* stands for Multipurpose Internet Mail Extensions and is an internet standard for transmitting text and attachments over standard protocols (in our case, SMTP). The method `sendExtentReport(String content, String subject)` uses two `MimeBodyPart` objects: one for the content and the other for the attachment, which is ExtentHtml.html.

So now we should modify our Cucumber `runner` class to send the extent report once it is fully cooked! The updated code for the `runner` class is provided in Listing A-4.

Listing A-4. AboutAppTest.java

```
package com.taf.testautomation.cucumber.aboutapp;

import com.taf.testautomation.utilities.emailutil.EmailUtil;
import com.taf.testautomation.utilities.fileutil.FileUtil;
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.AfterClass;
import org.junit.runner.RunWith;

import java.io.File;

import static com.taf.testautomation.utilities.excelutil.ExcelU
til.getCustomProperties;

@RunWith(Cucumber.class)
@CucumberOptions(features = "src/test/resources/features", glue
= {"com.taf.testautomation.cucumber.aboutapp"}, monochrome =
true, plugin = {"html:build/cucumber-html-report-normal",
        "json:build/cucumber.json", "com.taf.testautomation.
        cucumber.ExtentCucumberAdapter:"}, tags =
        {"@AboutApp"})
public class AboutAppTest {
    @AfterClass
    public static void sendExtentReport() {
        String folder = new Object() {
        }.getClass().getName();
```

```
        folder = folder.substring(folder.lastIndexOf('.') + 1,
        folder.indexOf('$'));
        EmailUtil.sendExtentReport("Extent Report is
        attached.", "Extent Report for last run");
        String htmlFilePath = getCustomProperties().
        get("reportPrefix") + "test-result/htmlreport/" + folder;
        String htmlFile = htmlFilePath + "/ExtentHtml.html";
        String folderExists = new File(htmlFilePath).mkdir() ?
        "Folder Created" : "Folder Exists";
        FileUtil.fileCopy("test-result/htmlreport/ExtentHtml.
        html", new File(htmlFile).getPath());
    }
}
```

So now our utilities folder looks like Figure A-1.

Figure A-1. *Utilities folder in Project Explorer*

APPENDIX B

Automation Setup

You may already be familiar with setting up tools for Android and iOS automation. However, as a reference, I put some notes for you here. Automation setup should be done in the following nine steps.

Step 1: Install Open JDK and Configure JAVA_HOME

To install openjdk on a Mac, first install Homebrew by running:

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/
Homebrew/install/master/install)"
```

Then, install openjdk by running `brew install openjdk`.

To update Homebrew at any time, run `brew update`.

If you want to run a specific version of openjdk, use `@version`. Example: `brew install openjdk@8`

Step 2: Install Gradle and Configure GRADLE_HOME

Install Gradle by running `brew install gradle`. For other installation options, please refer to https://gradle.org/install/.

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0

Step 3: Install Git and Set Up git config

Install Git by running `brew install git`.

Set up git config as per your organization's git host credential.

Step 4: Install Appium

Install Appium through npm.

Install npm by running `brew install node`.

If overwriting an existing node, run `brew link --overwrite node`.

To check for npm version, run `npm --version`.

Install Appium by running `npm install -g appium`.

To check for Appium version, run `appium --version`.

In iOS, there is a package named appium-doctor that helps you find out if your installation is complete.

To install appium-doctor, run `npm install appium-doctor -g`.

To check if installation is complete, run `appium-doctor –ios` and follow the instructions.

Step 5: Install Carthage

Carthage is a dependency manager for WebDriverAgent. Install Carthage by running `brew install carthage`.

Step 6: Install Xcode

Xcode is required for building the WebDriverAgent.xcodeproj that comes with Appium. Install Xcode from your app store.

If multiple versions of Xcode exist, then switch to the correct one using `xcode-select -switch`:

```
sudo xcode-select -switch /Applications/Xcode.app/Contents/
Developer/
```

For iOS automation, you have to run the `xcodebuild` command as follows:

```
xcodebuild -project WebDriverAgent.xcodeproj -scheme WebDriver
AgentRunner -destination 'id=xxxx' -allowProvisioningUpdates
test
```

If the WebDriverAgent in the iOS device is not trusted yet, running `xcodebuild` the first time will fail. Subsequently, you have to enable the trust setting for this app and run `xcodebuild` again.

Step 7: Install Android Studio

Install Android Studio by running `brew cask install android-studio`.

Step 8: Save System Variables in Bash Profile

Open `bash_profile` by running `open -e .bash_profile`.

Set up your `ANDROID_SDK`, `ANDROID_HOME`, `JAVA_HOME`, `GRADLE_HOME`, and `PATH` variables.

Step 9: Install IntelliJ

Install the community edition (free version) by running `brew cask install intellij-idea-ce`.

Index

A

K. Das, *Create an Enterprise-Level Test Automation Framework with Appium*,
https://doi.org/10.1007/978-1-4842-8197-0

B

C

D

E

F

G

H

I

J

K

L

M, N

O

P

Q

R

S

T

U, V

W, X, Y, Z