



C o m m u n i t y E x p e r i e n c e D i s t i l l e d

Robotium Automated Testing for Android

Efficiently automate test cases for Android applications using Robotium

Hrushikesh Zadgaonkar

[PACKT] open source*
PUBLISHING community experience distilled

www.allitebooks.com

Robotium Automated Testing for Android

Efficiently automate test cases for Android applications
using Robotium

Hrushikesh Zadgaonkar



BIRMINGHAM - MUMBAI

Robotium Automated Testing for Android

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: November 2013

Production Reference: 1141113

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78216-801-0

www.packtpub.com

Cover Image by Ravaji Babu (ravaji_babu@outlook.com)

Credits

Author

Hrushikesh Zadgaonkar

Project Coordinator

Sherin Padayatty

Reviewers

Krzysztof Fonał

Kristiono Setyadi

Michał Szpruta

Proofreader

Linda Morris

Indexer

Tejal Soni

Acquisition Editors

Kevin Colaco

Aarthi Kumaraswamy

Graphics

Abhinash Sahu

Commissioning Editor

Priyanka S

Production Coordinator

Shantanu Zagade

Technical Editor

Faisal Siddiqui

Cover Work

Shantanu Zagade

Copy Editors

Alisha Aranha

Kirti Pai

About the Author

Hrushikesh Zadgaonkar is a Software Developer by profession. He is currently working at Persistent Systems Limited, Nagpur, in the Telecommunication Business Unit. He studied engineering and was a Microsoft Student Partner and IBM Campus Ambassador in his college. He has been actively working on distinct domains such as .NET, Android, and the Robotium Framework. He was appointed as a NetBeans Certified Associate by the Oracle Corporation. Hrushikesh is a semi-finalist for the Imagine Cup 2010, a global competition organized annually by Microsoft. His research paper was selected for presentation at the CSE-IT International Level Conference 2010, Thailand.

He has been actively participating in various technical events in different colleges in Nagpur. His leisure activities include portrait sketching, playing the tabla, guitar, and sports such as cricket, football, and snooker. He is fond of social networking and appreciates innovation. He is a quick learner. Hrushikesh is an uncompromising fan of Sachin Tendulkar and Manchester United Football Club. His music interests lie with Enrique Iglesias, Bryan Adams, and A.R. Rahman.

When he isn't coding, he likes to hang out with his family and friends. He finds time every day to workout at the gymnasium. He is popularly called "Mr. Z" among his colleagues. He currently lives in Nagpur, India with his parents.

You can mail him at hzadgaonkar@gmail.com and he can be found tweeting at @MsWizKid.

I wish to thank my Mother who have always guided and supported me throughout my life and made me capable!

About the Reviewers

Krzysztof Fonał was born for programming. He started by trying to write a simple game on Commodore 64 at the age of 11. Between the ages of 13 and 16, he wrote a series of Ski Jump Manager games in Delphi (there were at least a 1,000 downloads). In December 2011, he graduated from Wrocław University of Technology, having at that time 1.5 years of experience in commercial .NET development (in PGS Software). In January 2012, he started work at Bitbar, a company which makes mobile test automation tools that are used by the biggest companies in the world. His office is not only the place when he develops. Recently, he launched his first Android game (Air Hockey), hoping there will be more of his titles on Google Play. Other than development, he likes sports (he won a few medals for powerlifting at Poland's tournaments), movies, and computer games.

Other books he might work on are about Android, Java, and Jenkins.

I'd like to thank my wife for having patiently lived with a nerd.

Michał Szpruta is a Software Engineer at Bitbar. He works at a group of projects called Testdroid. Michał has got experience with Robotium because of developing the Eclipse plugin called Testdroid Recorder for recording user actions, and generating reusable test cases (written in Robotium). He is the co-author of the library, which extends Robotium-recorder extensions with the main class ExtSolo, which can be found under: http://docs.testdroid.com/_pages/extsolo.html.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface	1
Chapter 1: Getting Started with Robotium	5
Robotium framework	5
Features and benefits	7
Setting up an Android environment	9
Requirements	9
Downloading the Android SDK	9
Installing ADT	10
Adding the Android SDK location	12
Installing the latest SDK version	13
Setting up the AVD	14
Summary	15
Chapter 2: Creating a Test Project Using Robotium	17
Creating the AUT	17
Creating a test project	26
Creating a test case	28
Adding the Robotium library	30
Adding the package name in AndroidManifest.xml	32
Robotium's test case code	33
Running the test case	35
Summary	36
Chapter 3: Robotium APIs	37
Solo	37
API calls	38
Resource ID in Robotium	39
Understanding internationalization	40
Summary	44

Chapter 4: Web Support in Robotium	45
API set	45
Hybrid test example	48
Summary	52
Chapter 5: Comparison with Other Frameworks	53
MonkeyRunner	53
Robolectric	55
UI Automator	56
Calabash	57
Summary	57
Chapter 6: Remote Control in Robotium	59
Software Automation Framework Support	59
Working of a Remote Control for Android	60
Robotium Remote Control using SAFS	61
Summary	63
Chapter 7: Other Robotium Utilities	65
The RobotiumUtils class	65
API set	66
XPath API and syntax	66
Robotium for pre-installed applications	68
Test for only APK	70
Signature process	70
Summary	72
Chapter 8: Robotium with Maven	73
Automate Android app built with Maven	73
Maven features	74
Setting up Android SDK and ADT	74
Setting up the environment PATH for Android Tools	75
Build Android tests using Maven	76
Summary	77
Index	79

Preface

Automation testing on mobile devices has been around for a number of years, although it has really taken off with the advent of the Robotium Framework.

With the help of automating test cases, business components are extensively reused and help to execute complex test cases. Due to a number of different key features added to the Robotium Framework, it has been the world's leading Android test automation framework and most industry experts and professionals are using this framework for testing their Android business applications.

The main aim to bring this book into the market is to provide users with detailed knowledge of the Robotium Framework and its features. After reading it, you should be good to go and create the automated test cases and run them for your Android project!

Welcome to Robotium automated testing for Android!

What this book covers

Chapter 1, Getting Started with Robotium, discusses the Robotium Framework and helps us install and set up the Android environment on Windows in a step-by-step manner.

Chapter 2, Creating a Test Project Using Robotium, guides you through the creation of a test project and helps to run it using Eclipse.

Chapter 3, Robotium APIs, introduces you to the `Solo` class and information about the APIs present in the framework. It will also teach you about internationalization.

Chapter 4, Web Support in Robotium, briefs you about accessing the Web Elements in Android using web support in Robotium.

Chapter 5, Comparison with Other Frameworks, aims to provide a comparison between Robotium and other testing frameworks based on certain parameters.

Chapter 6, Remote Control in Robotium, introduces you to the Software Automation Framework Support and the working of the Remote Control in Android.

Chapter 7, Other Robotium Utilities, consists of various utilities present in the Robotium Framework. These utilities include the `RobotiumUtils` class, XPath usage, Robotium usage for the already installed Android applications, and the signature process involved during the application sign-unsign operation to perform tests.

Chapter 8, Robotium with Maven, briefs you on the Maven tool that helps you to attach an Android project to a build process. This chapter also explains the different configurations you need to use Robotium with Maven.

What you need for this book

For this book, you'll need to have either a Windows XP (or newer), Linux, or Mac OS X operating system.

You'll need to download and install the Android SDK and Eclipse IDE (refer to the *Setting up an Android Environment* section in *Chapter 1, Getting Started with Robotium*).

Who this book is for

Robotium is a framework for automated test case developers for Android applications. This book aims to help beginners get acquainted with the Robotium SDK. You'll need some basic understanding on Java and Android programming and basic command-line familiarity.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "We can include other contexts through the use of the `include` directive."

Any website references are shown as follows:

`https://github.com/jayway/robotium/tree/master/robotium-solo`

A block of code is set as follows:

```
Activity activity = solo.getCurrentActivity();

ImageView imageView = (ImageView)
    solo.getView(act.getResources().getIdentifier("appicon", "id",
        act.getPackageName()));
```

Any command-line input or output is written as follows:

```
# adb push app.apk <path>
```

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Getting Started with Robotium

Automated testing helps us to maintain high software quality and provides a facility to capture if any code changes affect the actual use case. This chapter introduces the Robotium framework, its different features, and its benefits in the world of automated testing. By the end of this chapter, we will have a complete setup of the Android Environment in Eclipse IDE to get started with Robotium.

Robotium framework

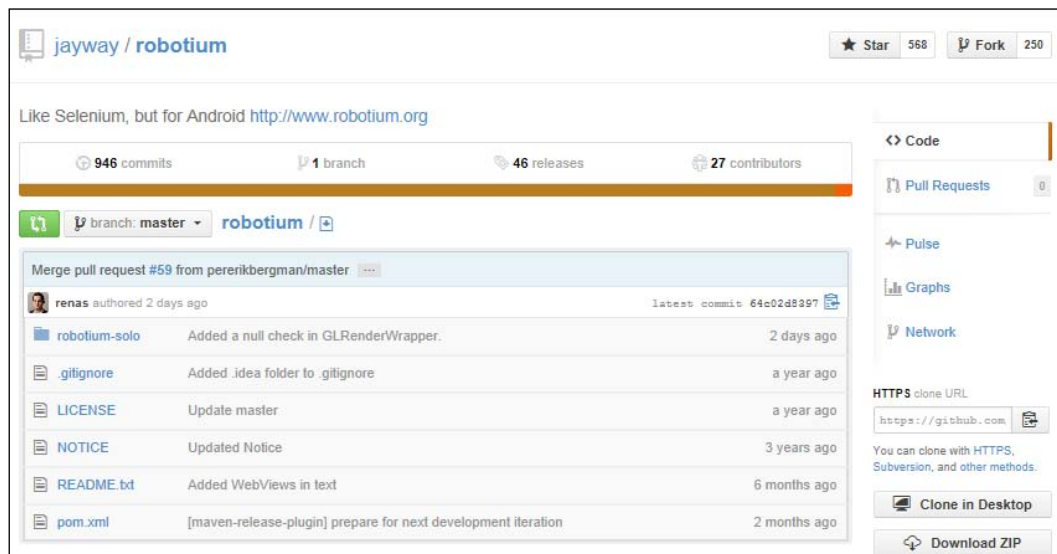
Robotium is an open source automation testing framework that is used to write a robust and powerful black box for Android applications (the emphasis is mostly on black box test cases). It fully supports testing for native and hybrid applications. Native apps are live on the device, that is, designed for a specific platform and can be installed from the Google Play Store, whereas Hybrid apps are partly native and partly web apps. These can also be installed from the app store, but require the HTML to be rendered in the browser.

Robotium is mostly used to automate UI test cases and internally uses run-time binding to **Graphical User Interface (GUI)** components.

Robotium is released under the Apache License 2.0. It is free to download and can be easily used by individuals and enterprises and is built on Java and JUnit 3. It will be more appropriate to call Robotium an extension of the Android Test Unit Framework, available at http://developer.android.com/tools/testing/testing_android.html. Robotium can also work without the application, under the test's source code.

The test cases written using Robotium can either be executed on the Android Emulator (**Android Virtual Device (AVD)**) – we will see how to create an AVD during installation in the following section – or on a real Android device. Developers can write function, system, and acceptance test scenarios across multiple activities.

It is currently the world's leading Automation Testing Framework, and many open source developers are contributing to introduce more and more exciting features in subsequent releases. The following screenshot is of the git repository website for the Robotium project:



As Robotium is an open source project, anyone can contribute for the purpose of development and help in enhancing the framework with many more features. The Robotium source code is maintained at GitHub and can be accessed using the following link:

<https://github.com/jayway/robotium>

You just need to fork the project. Make all your changes in a clone project and click on **Pull Request** on your repository to tell core team members which changes to bring in. If you are new to the git environment, you can refer to the GitHub tutorial at the following link:

<https://help.github.com/>

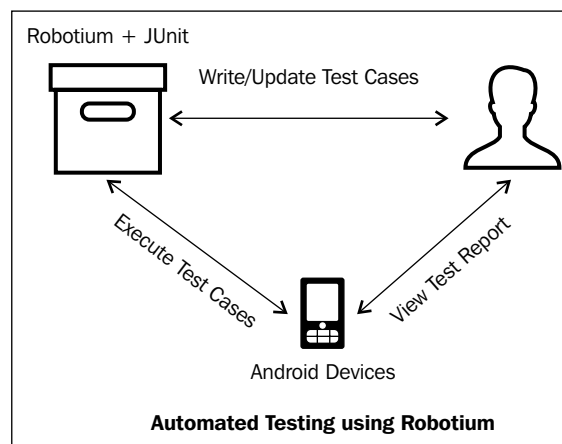
Robotium is like Selenium but for Android. This project was started in January 2010 by *Renas Reda*. He is the founder and main developer for Robotium. The project initiated with v1.0 and continues to be followed up with new releases due to new requirements. It has support for Android features such as activities, toasts, menus, context menus, web views, and remote controls.



Let's see most of the Robotium features and benefits for Android test case developers.

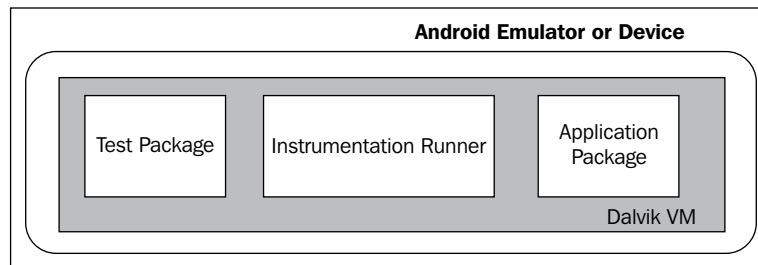
Features and benefits

Automated testing using Robotium has many features and benefits. The triangularization workflow diagram between the user, Robotium, and the Android device clearly explains use cases between them:



The features and benefits of Robotium are as follows:

- Robotium helps us to quickly write powerful test cases with minimal knowledge of the application under test.
- Robotium offers APIs to directly interact with UI controls within the Android application such as EditText, TextView, and Button.
- Robotium officially supports Android 1.6 and above versions.
- The Android platform is not modified by Robotium.
- The Robotium test can also be executed using command prompt.
- Robotium can be integrated smoothly with Maven or Ant. This helps to add Robotium to your project's build automation process.
- Screenshots can be captured in Robotium (an example screenshot is shown as follows):



- The test application project and the application project run on the same JVM, that is, **Dalvik Virtual Machine (DVM)**.
- It's possible to run Robotium without a source code.
- Robotium can work with other code coverage measurement tools, such as Cobertura and Emma.
- Robotium can detect the messages that are shown on the screen (Toasts).
- Robotium supports Android features such as activities, menu, and context menu.
- Robotium automated tests can be implemented quickly. Robotium is built on JUnit, because of which it inherits all JUnit's features. The Robotium framework automatically handles multiple activities in an Android application.
- Robotium test cases are prominently readable, in comparison to standard instrumentation tests.
- Scrolling activity is automatically handled by the Robotium framework.

- Recent versions of Robotium support hybrid applications. Hybrid applications use WebViews to present the HTML and JavaScript files in full screen, using the native browser rendering engine.

Setting up an Android environment

You can set up an Android environment in Eclipse, which is the primary step to use Robotium for creating a test project, as follows:

Requirements

Before the actual setup of the Android environment for Robotium, you will need to check all the following required elements:

- The **Java Development Kit (JDK)** must be installed (you can install it from <http://www.oracle.com/technetwork/java/javase/downloads/index.html>)
- Eclipse IDE must be installed
 - Standard Eclipse IDE (<http://www.eclipse.org/downloads/>)
 - Eclipse IDE with built-in **Android Developer Tools (ADT)** (<http://developer.android.com/sdk/index.html>)
 - For Robotium for Android Starter, we will use Standard Eclipse IDE, which is mostly used by tech enthusiasts and developers across industries. Eclipse IDE with built-in ADT has an Android plugin and there is no need to set up the Android SDK. Only one of the two Eclipse IDEs is needed.
 - To use standard Eclipse IDE for Android Development and setup a new SDK, you need to download SDK tools and select the additional SDK packages to install. In the existing version of Eclipse IDE, add ADT plugin

Downloading the Android SDK

The easiest way to download the Android SDK is by getting a compressed ADT package from <http://developer.android.com/sdk/index.html>.

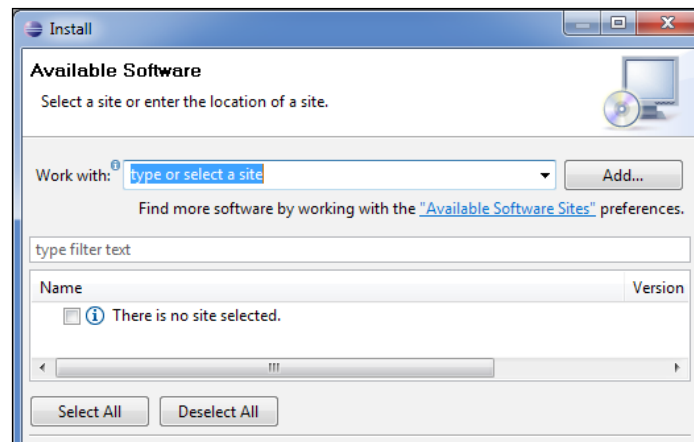
The Android SDK provides libraries and developer tools to build, test, and debug Android applications.

Unpack it to a safe location on your system. We will use this package in the following steps.

Installing ADT

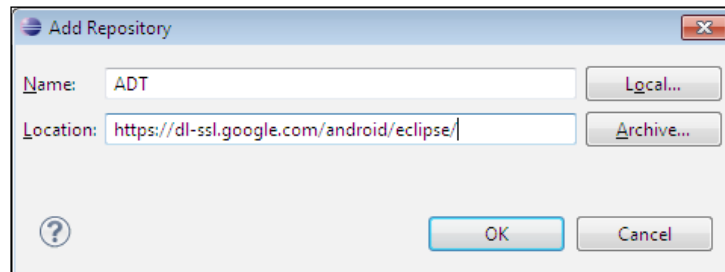
You can install ADT by following the listed steps:

1. In Eclipse IDE, Kepler, click on the **Help** menu and then on the **Install New Software** option. You will get the following screen, which shows the available software depending on the website URL provided in the **Work with:** combo box. The **Install New Software** wizard allows you to add new software to your installation, as shown in the following screenshot:

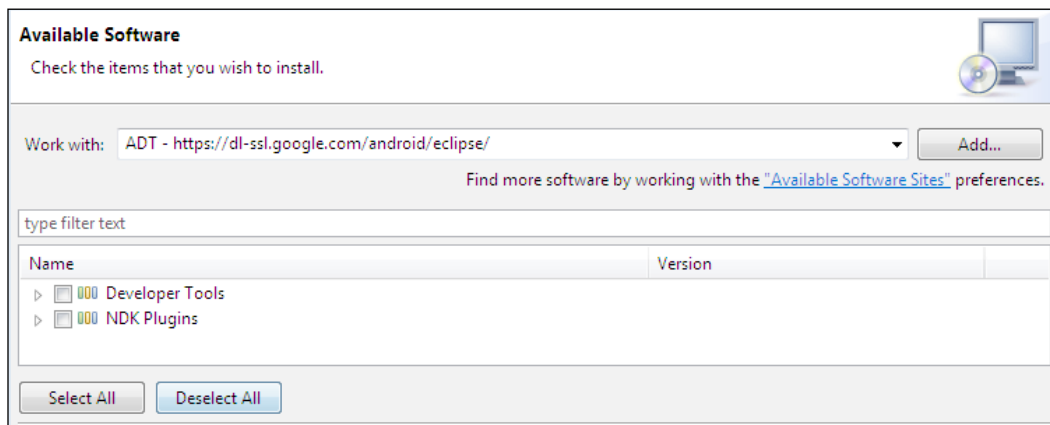


2. Using the **Work with:** combo box, you can always select any website to browse its contents. You can also browse through all the software from these sites. This is useful when you know the software name, but not the actual location.

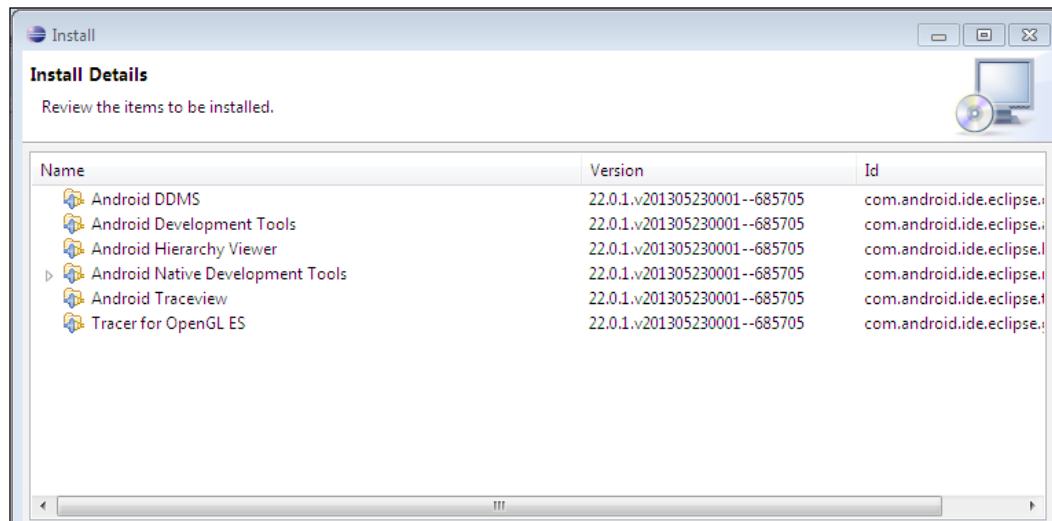
3. Click on the **Add** button in the **Install New Software** window that will open the **Add Repository** window, which looks like the following screenshot.
4. In this window, enter a name in the **Name** field and the following URL in the **Location** field and click on the **Ok** button to download the Android ADT:



5. Eclipse will now search for all the available tools from this location and enlist them as follows:



6. Select all the tools and click on **Next**. This will open up a window, shown in the following screenshot, with a list of all the components that will be installed as a plugin in Eclipse:

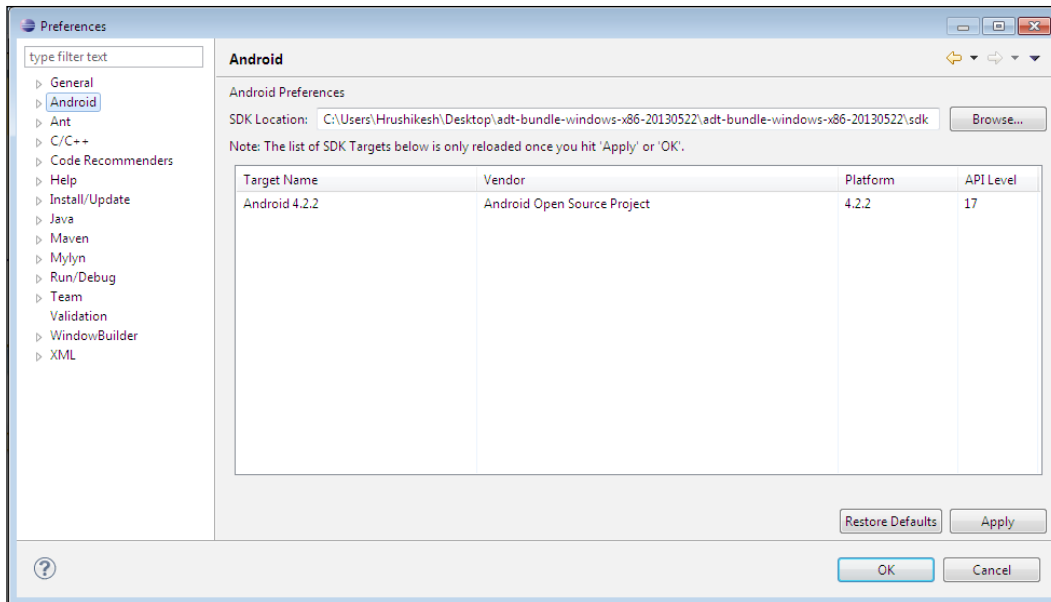


7. Click on the **Next** button present in the **Install Details** window. It will start downloading all the mentioned tools after the license verification is done. After successful installation, you will be asked to restart Eclipse IDE. Restart it!

Adding the Android SDK location

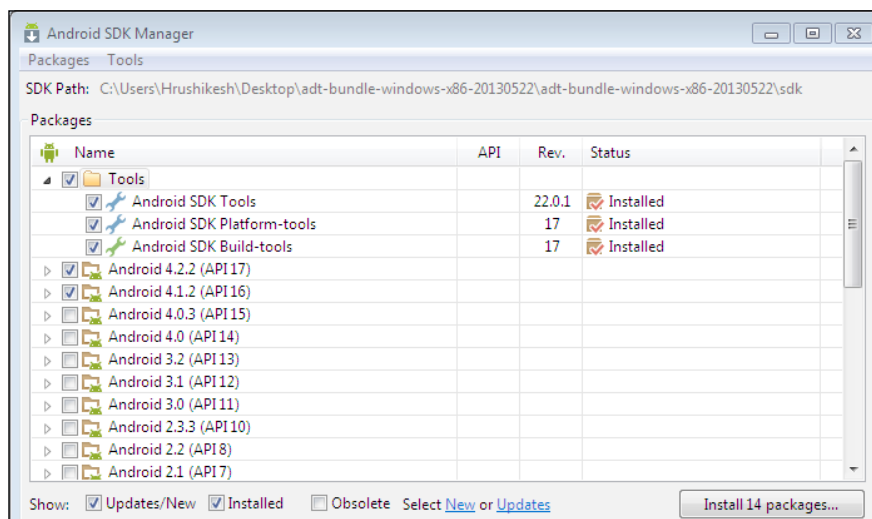
To add the Android SDK to Eclipse, follow the listed steps:

1. In Eclipse, under the **Window** menu, click on **Preferences** (Mac OS X users may find **Preferences** under the **Eclipse** menu). Check the **Android** tab (The presence of this tab clearly indicates that the Android ADT plugin is successfully installed.) and a window, as shown in the following screenshot, will appear. The SDK location informs Eclipse where the Android SDK is located in the system.
2. If the SDK location is not present, browse to the SDK unzipped directory and click on **OK**. The list of SDK targets will be reloaded only when the proper SDK location is provided and the **Apply** or **OK** button is clicked on. These targets are taken from the SDK itself.
3. If you still don't get any entry in the list of targets, this means your Android SDK is not installed properly. Install the Android SDK as mentioned in step 3 and check for the SDK target in the list:



Installing the latest SDK version

Before actually creating the virtual device, you need to install the latest version of SDK. Go to **Android SDK Manager** from the **Window** menu, and a window, as shown in the following screenshot, will appear. Select the latest version of the SDK installed. You can select any version based on your choice and click on **Install Packages....** Once installed, restart Eclipse if the package's installation changes are not reflected:

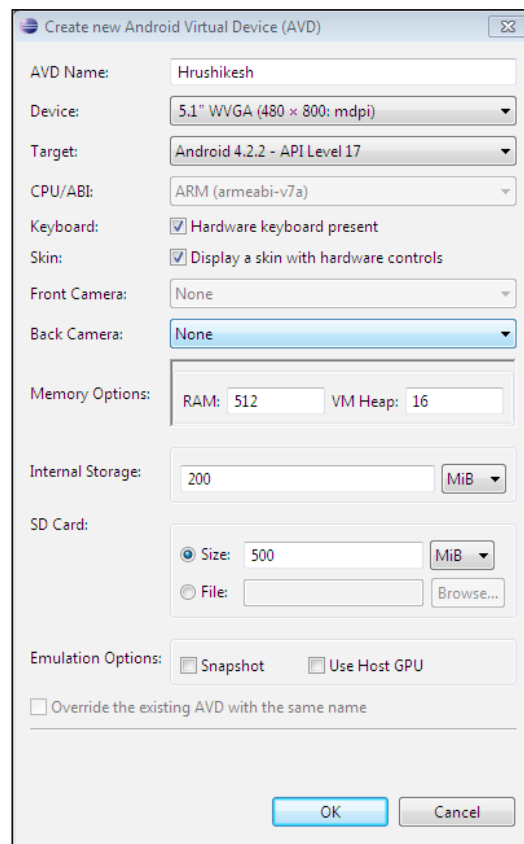


Setting up the AVD

The initial configuration of the Android Environment is almost done. Now, we are only left with the set up of the AVD.

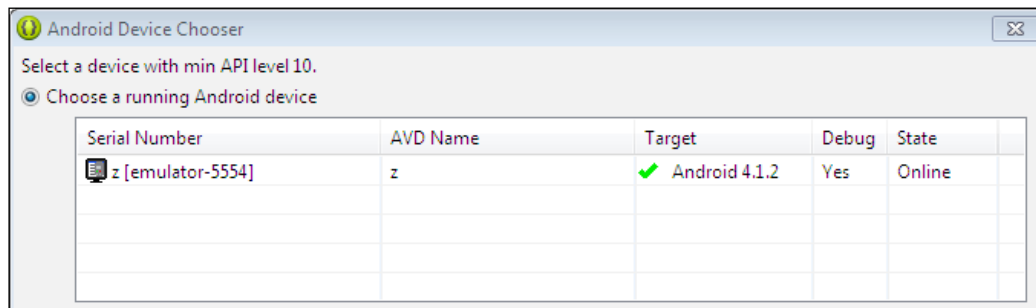
An AVD is used to run Android applications. It is recommended that you use an Android device to run applications. But within the scope of this book, we will only use AVD (Android Emulator) to run apps.

You can create a new AVD from the **AVD Manager** option, present under the **Window** menu in Eclipse. Click on **New** in the AVD screen and you will see a window like the following screenshot. Fill in the following details and click on **OK**.



Once it is created, it is shown under the Android SDK and the AVD manager screen.

To run the created AVD, right-click on the project and navigate to **Run As | Android Application**. A **Deployment Target Selection Mode** window will pop up, asking you to select an AVD or a connected Android device to run your application; select either one of them and the application gets installed on the selected device/AVD. The following screenshot is of the **Deployment Target Selection Mode** window:



By this point, you should have a working setup of the Android Environment to start with Robotium's test case writing and execution. You are free to play around and discover more about it.

Summary

In this chapter, we've learnt about the Robotium framework and what are the different steps you need to perform to make your Android Environment ready to get started with the framework.

In the next chapter, we'll start implementing our first test project using Robotium. So, keep reading if you want to learn about test case implementation.

2

Creating a Test Project Using Robotium

This chapter will guide you in creating your first test project for Android using the Robotium Framework. First, let's implement a simple calculator android application. Then, using this **Application Under Test (AUT)**, we will look into the process of creating a Robotium test project.

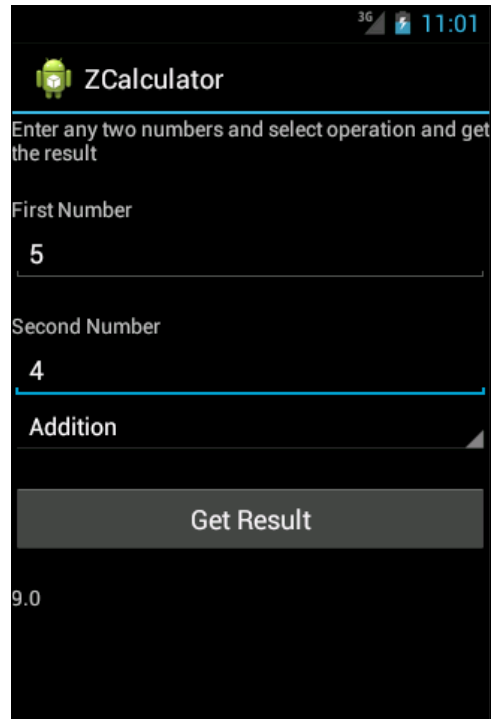
Creating the AUT

In this section, we will create a simple calculator application that allows the entry of two numbers. The user can perform the following two operations on these numbers:

- Addition
- Subtraction

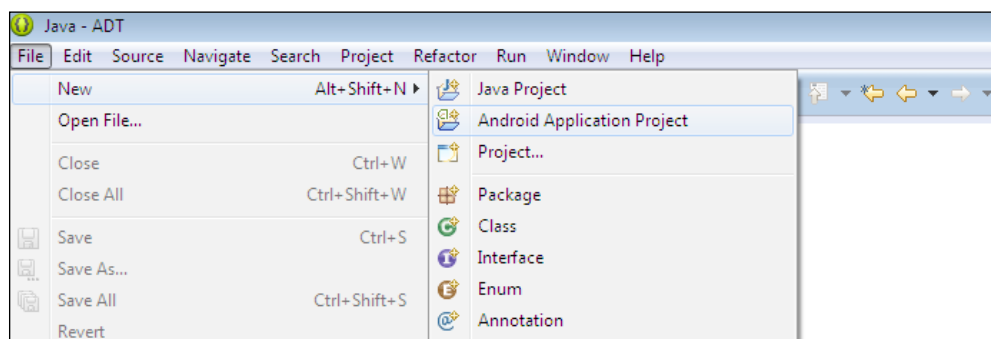
These operations are selectable via Spinner control. Spinner is similar to a combo box present in other coding languages such as HTML, and C#. The **Get Result** button is present to get the operational result in the bottom-aligned TextView.

The following screenshot shows the ZCalculator app:

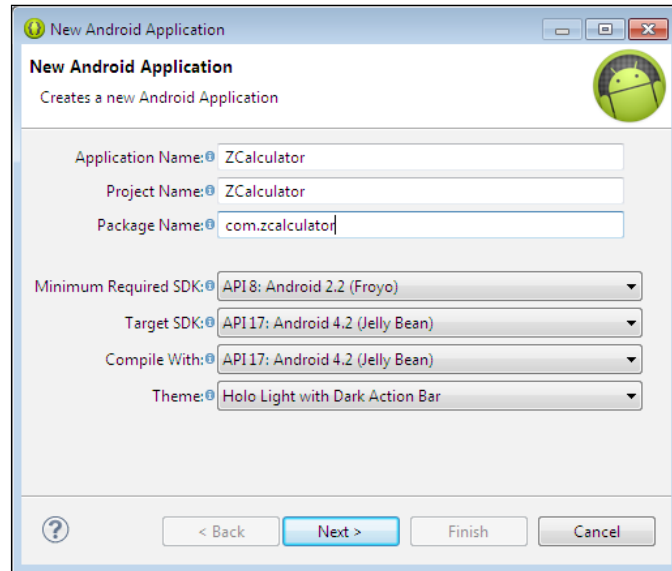


To create the AUT, follow the listed steps:

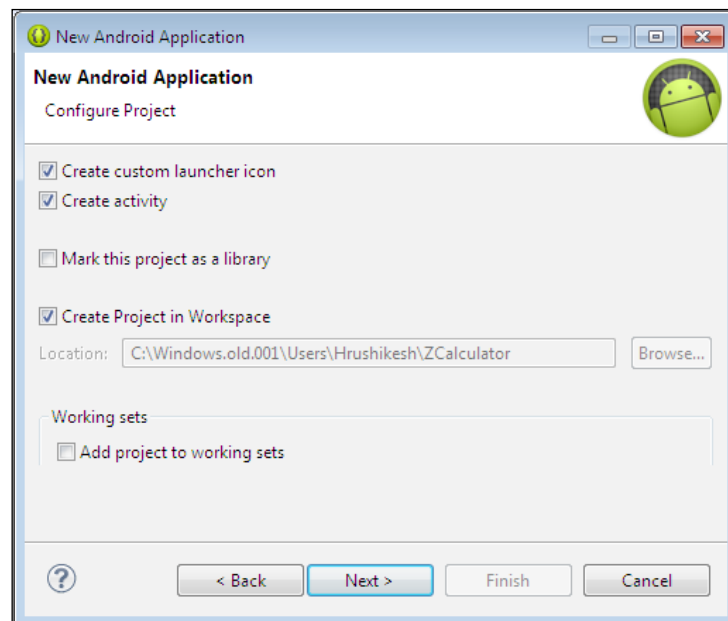
1. Create an **Android Application Project** by navigating to **File | New | Android Application Project** in Eclipse IDE.



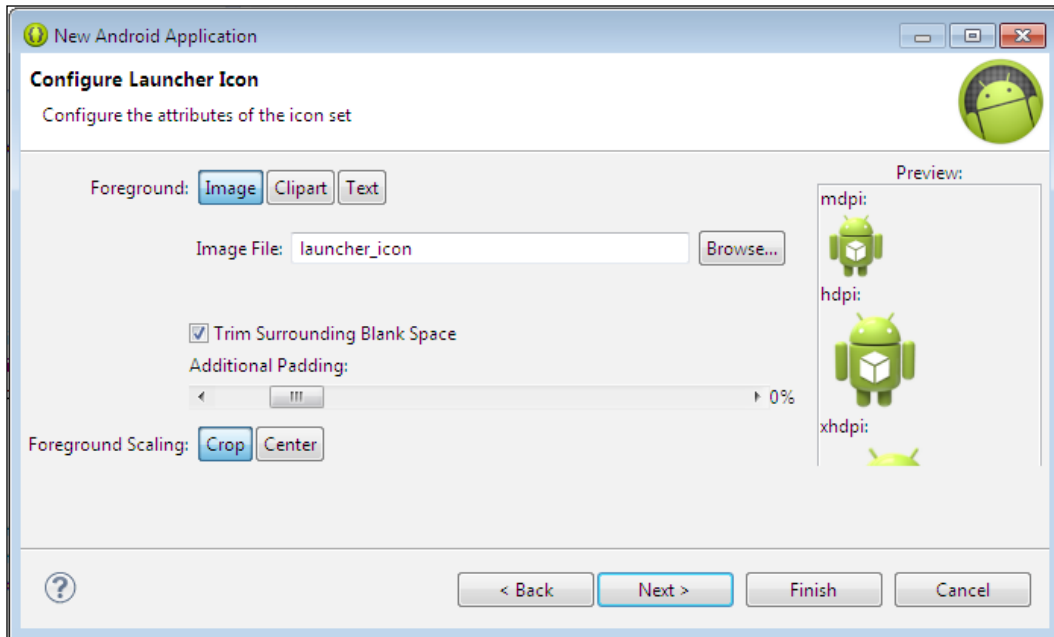
2. Enter the application details, as in the following screenshot, and click on the **Next** button:



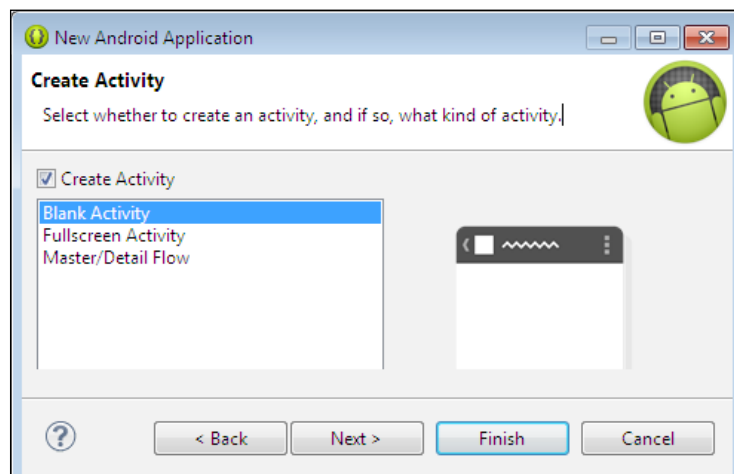
3. Keep the default options, as they are in the following screenshot, and click on the **Next** button:



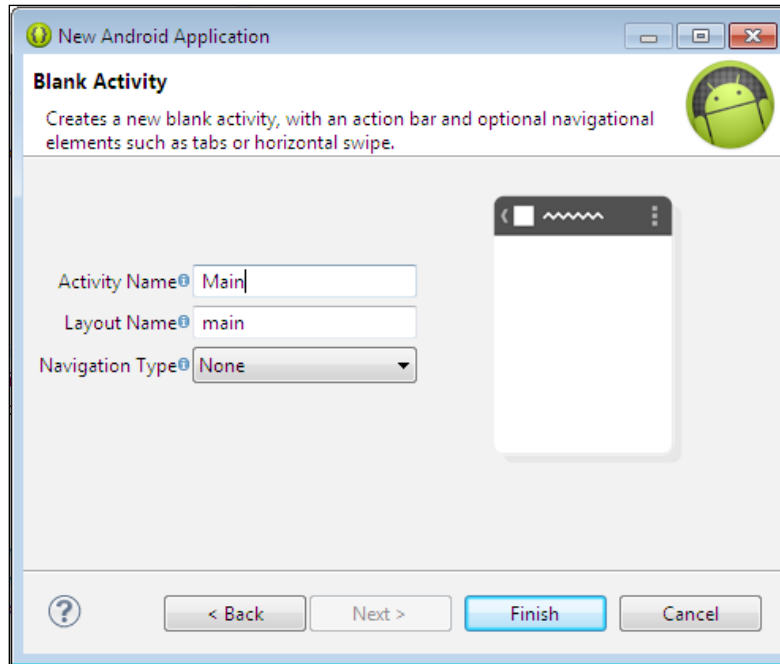
4. For this Android application project, we will configure the launcher icon with the default value set, as in the following screenshot, and click on the **Next** button:



5. Check the **Create Activity** checkbox if it is not checked and select **Blank Activity**, as in the following screenshot, to create a default blank activity class in the project:



6. Enter **Main** in the **Activity Name** field, as in the following screenshot, and click on the **Finish** button to create the Android application project:



Your Android project is now set up. You can refer to the ZCalculator project code base, given as follows:

1. Use the following code in your `Main.java` file:

```
package com.zcalculator;

import com.calculator.R;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.TextView;

public class Main extends Activity {
    Spinner    operationSpinner;
    TextView   result;
```

```
Button    getResult;

private enum OperationType
{
    Addition, Subtraction
}

@Override
public void onCreate(final Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    this setContentView(R.layout.main);

    final EditText txtfirstNumber = (EditText)
        this.findViewById(R.id.txtFirstNumber);
    final EditText txtsecondNumber = (EditText)
        this.findViewById(R.id.txtSecondNumber);

    this.result = (TextView) this.findViewById(
        R.id.resultText);
    this.result.setText("0.00");

    this.getResult = (Button) this.findViewById(
        R.id.btnGetResult);

    this.operationSpinner = (Spinner) this.findViewById(
        R.id.operationSpinner);

    // Adding listener to get result button
    this.getResult.setOnClickListener(new View.
        OnClickListener() {

        public void onClick(final View v) {
            OperationType operationType = OperationType.
                valueOf(Main.this.operationSpinner.
                    getSelectedItem().toString());

            final float num1 = Float.parseFloat(
                txtfirstNumber.getText().toString());
            final float num2 = Float.parseFloat(
                txtsecondNumber.getText().toString());

            // Getting first & second values and passing to
            show result
            Main.this.showResult(num1,num2 ,operationType);
        }
    }
}
```

```
    });  
}  
  
// Showing operation results  
protected void showResult(final float firstNumber,  
    final float secondNumber, final OperationType type) {  
  
    float resultVal = 0;  
    if (type.equals(OperationType.Addition)) {  
        resultVal = firstNumber + secondNumber;  
    } else if (type.equals(OperationType.Subtraction)) {  
        resultVal = firstNumber - secondNumber;  
    }  
  
    String operationResult = String.valueOf(resultVal);  
    this.result.setText(operationResult);  
}  
}
```

2. Use the following code in the `main.xml` layout file:

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
  
    <TextView  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:text="@string/hello"/>  
  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="@string/txtSpace"/>  
  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="@string/txtFirstNumber"/>  
  
    <EditText  
        android:inputType="numberDecimal"  
        android:id="@+id/txtFirstNumber"
```



```
        android:layout_width=" match_parent"
        android:layout_height="wrap_content"/>

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/txtSpace"/>

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/txtSecondNumber"/>

<EditText
    android:inputType="numberDecimal"
    android:id="@+id/txtSecondNumber"
    android:layout_width=" match_parent"
    android:layout_height="wrap_content"/>

<Spinner
    android:id="@+id/operationSpinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/spinnerItems"/>

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/txtSpace"/>

<Button
    android:text="@string/btnResultString"
    android:id="@+id/btnGetResult"
    android:layout_width=" match_parent"
    android:layout_height="wrap_content"/>

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/txtSpace"/>

<TextView
    android:id="@+id/resultText"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/txtSpace"/>

</LinearLayout>
```

3. Update the `String.xml` file with the following entries:

```
<string name="hello">Enter any two numbers and
    select operation and get the result</string>
<string name="app_name">ZCalculator</string>
<string name="txtFirstNumber">First Number</string>
<string name="txtSecondNumber">Second Number</string>
<string name="btnResultString">Get Result</string>
```

4. Update the `array.xml` file with the following entries:

```
<string-array name="spinnerItems">
    <item>Addition</item>
    <item>Subtraction</item>
</string-array>
```

5. Also, update the `AndroidManifest.xml` file with the following activity action and launcher entries:

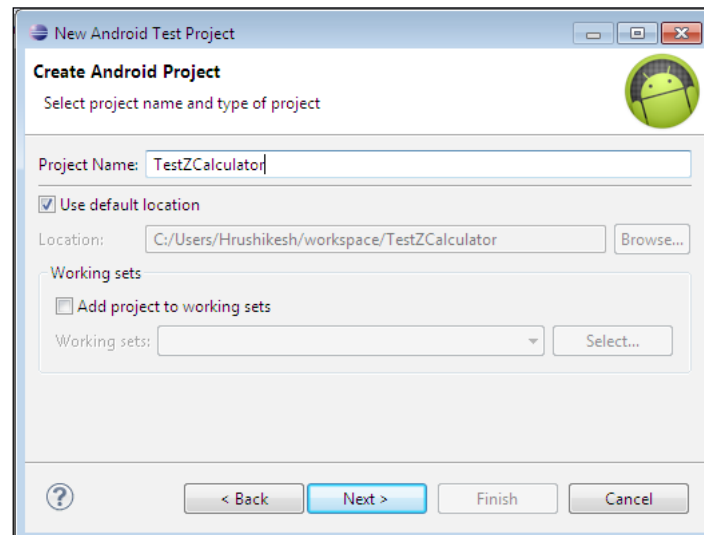
```
<uses-sdk android:minSdkVersion="8"/>

<application android:icon="@drawable/ic_launcher"
    android:label="@string/app_name">
    <activity android:name="com.zcalculator.Main"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="
                android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

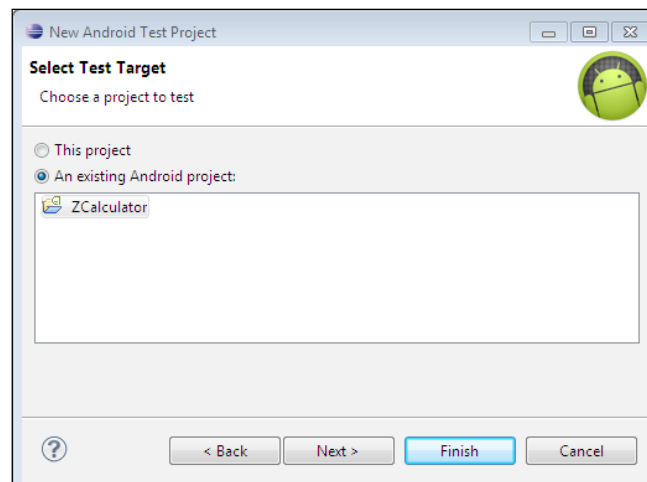
Creating a test project

Let's proceed and create a test project to test the ZCalculator application.

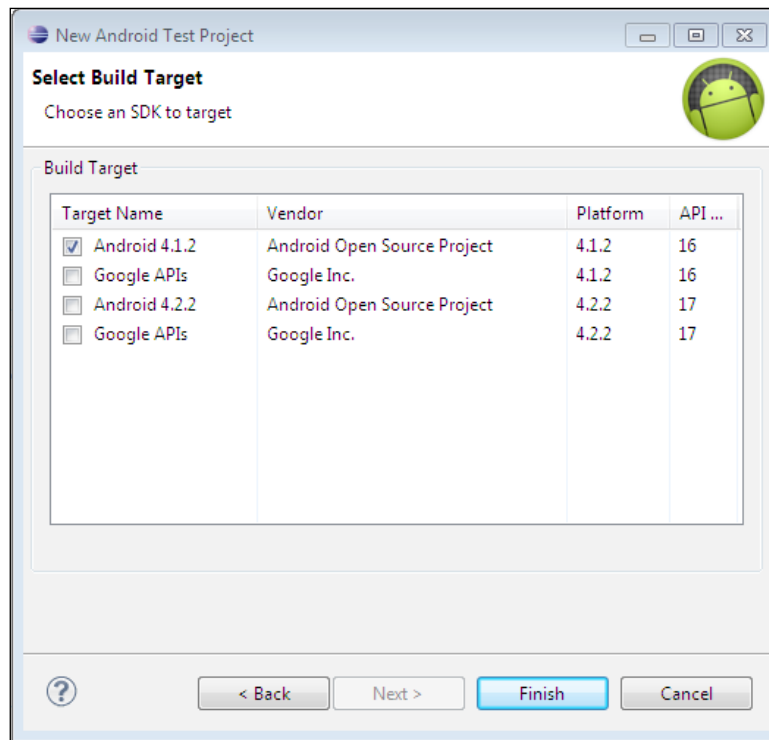
In Eclipse, go to **New** and from the **Select** wizard, select **Android Test Project**. Enter a proper project name and click on the **Next** button. It is recommended that the test project name follow a naming convention such as "Test + AUT name." That's why this test app is named `TestZCalculator`, as shown in the following screenshot:



Then, select the AUT or the target project (in our case, ZCalculator), as in the following screenshot, and click on the **Finish** button:



Select a build target, as shown in the following screenshot, (SDK to target) and click on the **Finish** button:

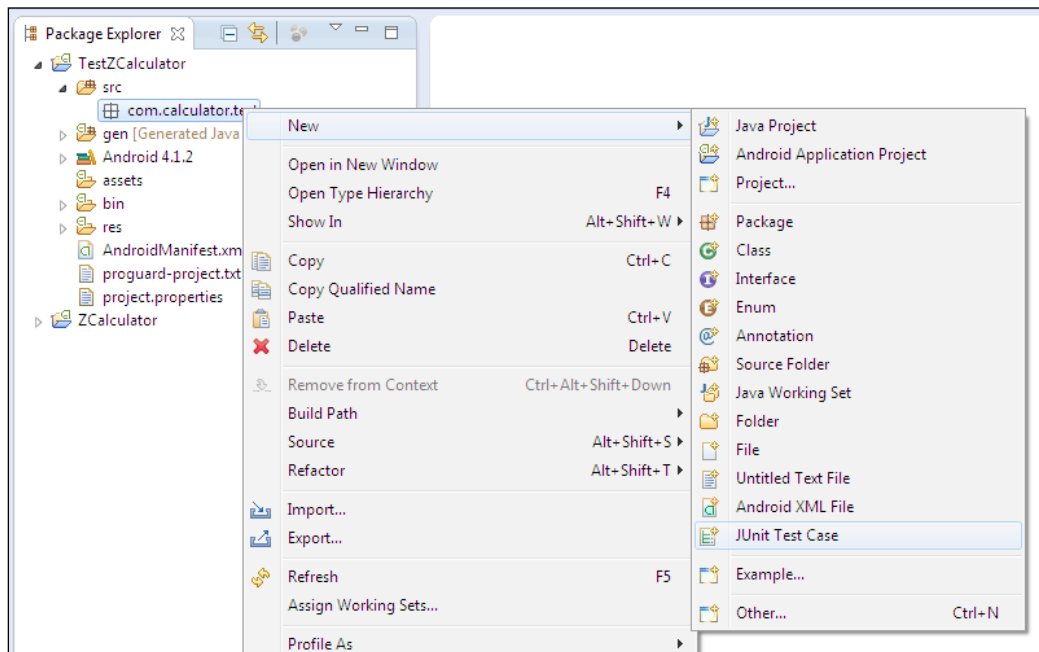


Your test project is successfully created. Let's create a test case class to test ZCalculator's main class.

Creating a test case

To create a test case, follow the listed steps:

1. To create a test case, right-click on the `com.calculator.test` package in the **Package Explorer** window and navigate to **New | JUnit Test Case**, as shown in the following screenshot. If this package is not present, create it under the **src** branch:



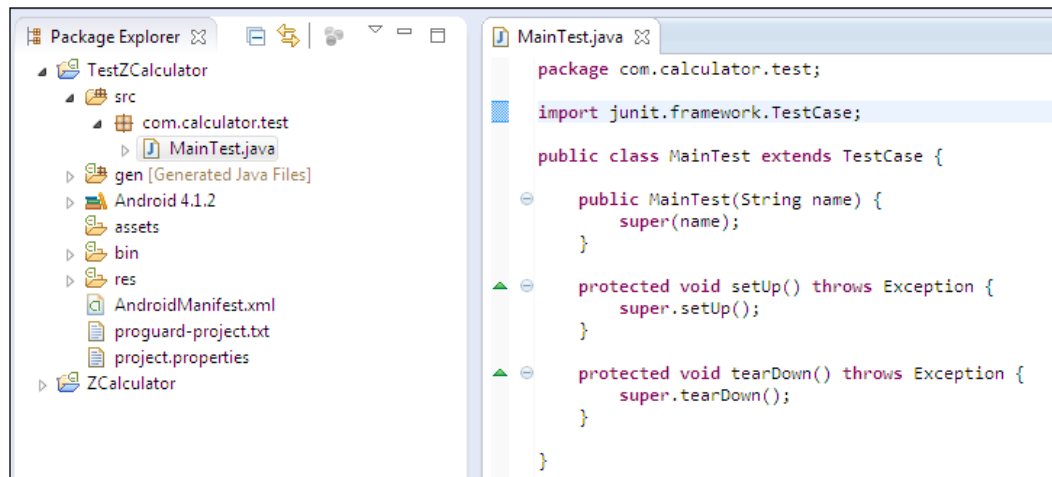
2. On the **New JUnit Test Case** window, most of the fields are already filled. Just assign the name of the test case as `MainTest`, as we are going to test the `Main` class in `ZCalculator`. Keep the **setUp()**, **tearDown()**, and the **constructor** option checkboxes checked in the method stubs section and click on the **Finish** button.



The `setUp()` and `tearDown()` methods are part of the `junit.framework.TestCase` class. The `setUp()` method is used to initialize the data needed to run the tests and reset environment variables. The `tearDown()` method is used to call the garbage collection to force the recovery of memory. It is called after each `@Test` method, as shown in the following code:

```
Call @Before setUp
Call @Test method test1
Call @After tearDown
Call @Before setUp
Call @Test method test2
Call @After tearDown
```

3. Once it is completed, a test case `MainTest.java` will be created under the `com.calculator.test` package, as shown in the following screenshot. Also, the three stub methods will be generated automatically in the `MainTest` class:

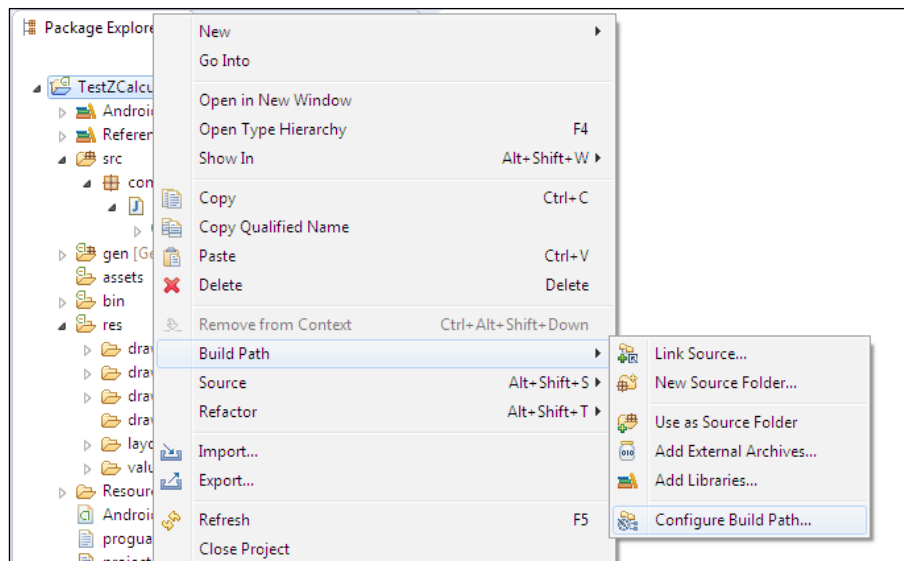


Adding the Robotium library

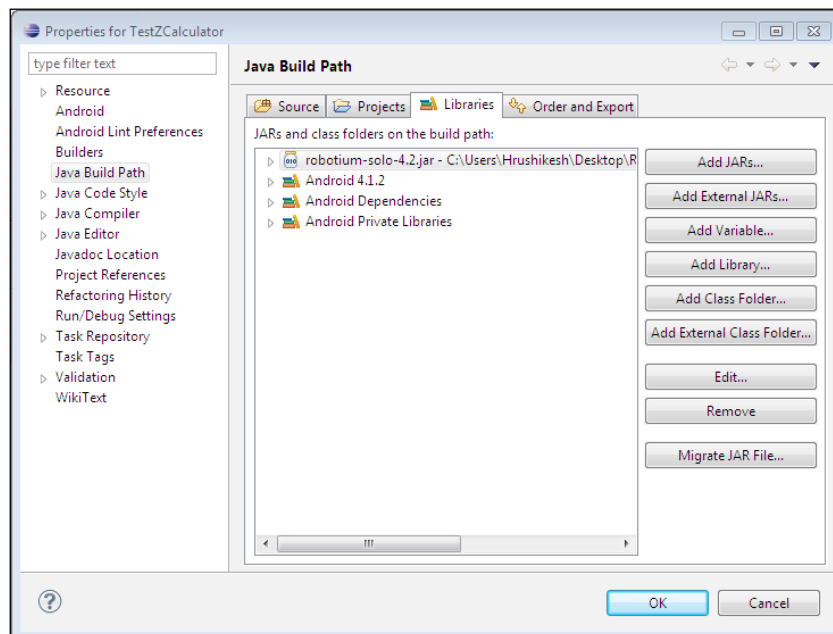
All versions of the Robotium JAR file can be downloaded from <https://code.google.com/p/robotium/downloads/list>.

The Robotium library can be added by following the listed steps:


1. You need to add the Robotium JAR as a reference library to the test project. To reference this, right-click on your project and navigate to **Build Path** | **Configure Build Path**, as shown in the following screenshot:

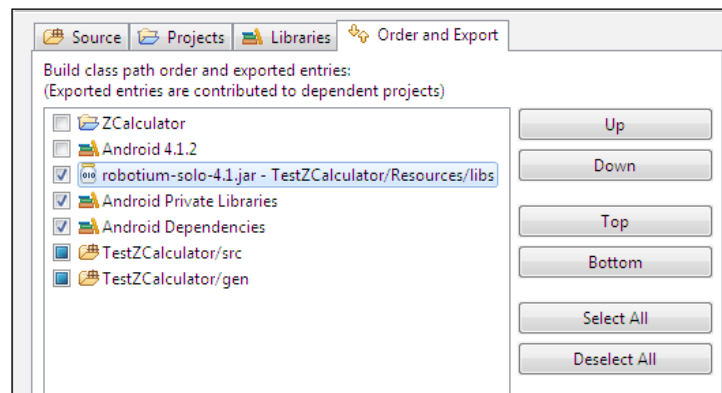


2. In the **Java Build Path** panel, go to the **Libraries** tab and click on the **Add External JARs...** button, as shown in the following screenshot. Then, provide a correct Robotium library (preferably the latest version) and add it to the list. The alternative way to achieve this is to copy the JAR file to the `lib` directory of the test:



3. It is mostly observed in the latest SDK versions (mostly API 17 and above) that the **java.lang.NoClassDefFoundError: com.jayway.android.robotium.solo.Solo** error occurs when the Robotium JAR file is not exported. So, to export it, go to the **Order and Export** tab in the **Java Build Path** section and check the Robotium JAR file in the list, as shown in the following screenshot, and click on the **OK** button:

[ It is important to have Android Private Libraries checked; otherwise the test won't start.]



Adding the package name in AndroidManifest.xml

Once you provide the reference of the Robotium library to the test project, open the `AndroidManifest.xml` file and change the target package name, as follows:

```
<instrumentation android:targetPackage="com.calculator"
    android:name="android.test.InstrumentationTestRunner" />
```

The following screenshot shows the preceding change in the `AndroidManifest.xml` file:



Robotium's test case code

Before going into the actual code, there are some classes and methods of the Robotium framework you should be familiar with.

`Solo` is the class of Robotium that is used for testing. It is initialized with the instrumentation of the test case and the first activity to test. This is performed in the `setUp()` method. The `Solo` class provides APIs with easy calling of Android UI components, for example, the `enterText()` API puts the text in an `EditText` view. We will see most of these APIs in the following section.

The test case method name in the JUnit should always start with the word "test." Since Robotium is built on JUnit, we have the `testZCalculatorBlackBox()` method for the test case. You can add any number of test case methods in a test case class.

In the following test case, we will access the UI components of `ZCalculator` and carry out the following operations in sequence:

1. Access the **Edit Text** field for inputs (first and second numbers).
2. Enter any value.
3. Access and click on **Spinner** to select the operation.
4. Access and click on the **Get Result** button.

Put the following code into the `MainTest.java` file and save it:

```
package com.zcalculator.test;

import android.test.ActivityInstrumentationTestCase2;
import com.jayway.android.robotium.solo.Solo;
import com.zcalculator.Main;

public class MainTest extends ActivityInstrumentationTestCase2<Main> {
    private Solo    solo;

    public MainTest() {
        super(Main.class);
    }

    @Override
    protected void setUp() throws Exception {
        super.setUp();
        this.solo = new Solo(this.getInstrumentation(),
            this.getActivity());
    }

    public void testZCalculatorBlackBox() {

        // Enter 5 in first number field
        this.solo.enterText(0, "5");

        // Enter 4 in second number field
        this.solo.enterText(1, "4");

        // Press Addition Spinner Item
        this.solo.pressSpinnerItem(0, 0);

        // Click on get result button
        this.solo.clickOnButton(0);

        // Verify that resultant of 5 + 4
        assertTrue(this.solo.searchText("9"));

        // Press Subtraction Spinner Item
        this.solo.pressSpinnerItem(0, 1);

        // Click on get result button
        this.solo.clickOnButton(0);

        // Verify that resultant of 5 - 4
        assertTrue(this.solo.searchText("1"));
    }
}
```

```

@Override
protected void tearDown() throws Exception {
    this.solo.finishOpenedActivities();
}
}

```

Running the test case

We are now done with creating a test project with a valid test case for ZCalculator. It's time to run our test case.

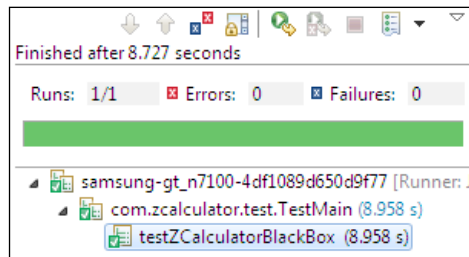
Right-click on the test project or test case file, `MainTest.java`, and select **Run as Android JUnit test**. Select **Android Emulator** from the select device screen.



If you want to run a particular test case, right-click on the file and then select **run as Android JUnit Test**. To run all the test cases available in the test project, right-click on the project itself and select **run as Android JUnit Test**, it will run all the test cases.

Robotium's test case for ZCalculator will work as follows:

1. The ZCalculator application will be loaded.
2. The first and the second numbers will be automatically entered in the first and second **Edit Text** fields, and then the spinner will be clicked on for selecting the operation (it will take Addition first).
3. The **Get Result** button will be clicked on and the result will be displayed in the result text view.
4. The assert statement will check for the valid operation result. This process will continue for Subtraction and if every assert is true, the test case is passed, indicated by a green bar in the JUnit tab, as depicted in the following screenshot:



The application and instrumentation must be installed beforehand, if you want to run the test project via the Command Line. If they are already installed, then use the following command:

```
adb shell am instrument -w com.calculator.test/  
    android.test.InstrumentationTestRunner
```

Before running the preceding command, note that you run it from the location where `adb.exe` is present, or add the `adb.exe` path to the environment path's variable list to access it from anywhere in the system.

You can find `adb` in the `platform-tools` folder inside the Android SDK.

Summary

In this chapter, we have seen how to create a test project using the Robotium Framework. Until now, you have learnt the basic flow required to create a simple test app using Robotium. It is time to go deeper into the framework and understand different Robotium API calls with their usage. In the next chapter, you will be introduced to the `Solo` class and information about APIs present in it.

3

Robotium APIs

This chapter will introduce you to the `Solo` class and information about the APIs present in the framework. Once this is completed, we will consider the Resource ID test case, which briefs us on how to achieve internationalization using Robotium.

By the end of this chapter, you will get to know most of the APIs in the framework and test case evaluation.

Solo

The `Solo` class is a main class provided in the Robotium framework, which consists of APIs to write test cases in a project. Robotium can be used in conjunction with Android test classes, such as `ActivityInstrumentationTestCase2` and `SingleLaunchActivityTestCase`.

The `Solo` class has the following two constructors:

- `Solo (android.app.Instrumentation instrumentation):`
This constructor takes in instrumentation as a parameter
- `Solo(android.app.Instrumentation instrumentation, android.app.Activity activity):` This constructor takes in instrumentation, as well as activity

API calls

There are many APIs present inside the Robotium framework and they cover most of the features available in Android. The method count increases, based on user feedback and suggestions. If any Robotium test case developer working on a test project finds that there are some methods (referring to a particular useful functionality) that can be added as a part of the Robotium framework, then it can help people in their respective projects.



The Robotium developer team analyzes these new requirements based on priority. Some of them are implemented and get added/deleted as a part of the next release. If any API support gets discontinued in the next release version, updating your test project Robotium lib could be troublesome.

All these methods are available online at the following links:

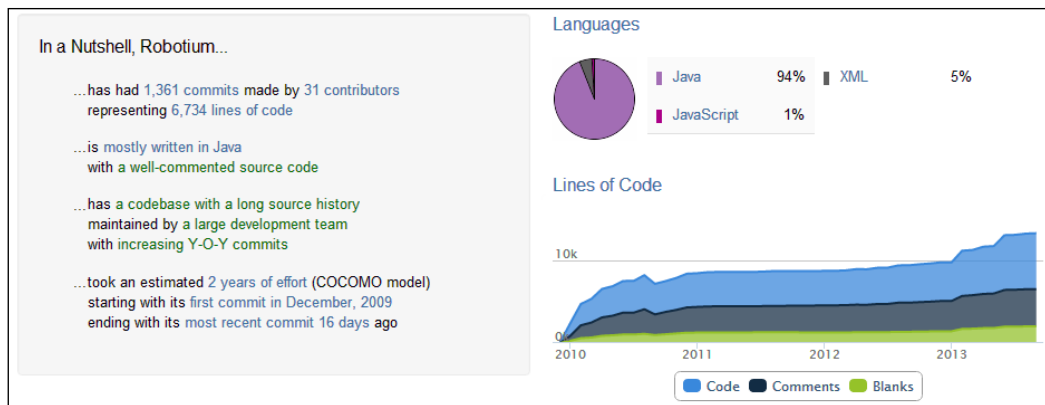
<http://robotium.googlecode.com/svn/doc/com/jayway/android/robotium/solo/Solo.html>

<http://robotium.googlecode.com/svn/doc/com/jayway/android/robotium/solo/Solo.html>

You can either study the Javadoc for the API set, or you can browse through the source code available at

<https://github.com/jayway/robotium/tree/master/robotium-solo>

The brief insight on Robotium, including the total commits, LOC, project model, and technology stack is represented in the following figure:



Snapshot from ohloh.net

Resource ID in Robotium

In Robotium, you don't need to import the **Application Under Test (AUT)** resources to the test project in order to use the Resource ID.



You can do it without importing the resource folder. All you need is to get the Resource ID by passing the name and type of the view to the `getIdentifier()` method and then passing the obtained ID to the `getView()` method to get the respective view object.

```
Activity activity = solo.getCurrentActivity();

ImageView imageView = (ImageView)
    solo.getView(act.getResources().getIdentifier(
        "appicon", "id", act.getPackageName()));
```

It works with the string parameter as well; for example:

```
Solo.getView("com.robotium.R.id.icon");
```


Understanding internationalization

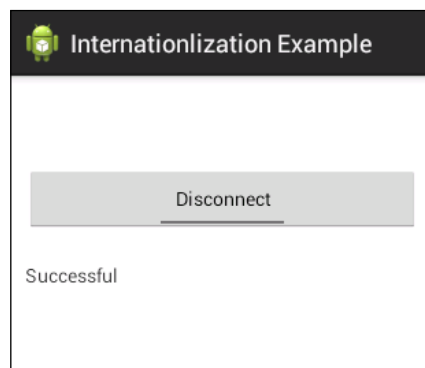


You might be wondering as to what internationalization is. Well, it is the means of adapting an application to various languages or regions. This might be confused with the concept of localization, but both concepts are slightly different. Localization means adapting the application to any region or language, thereby using locale components and translating the text.

Consider an application that can have multiple language support. If you want to test this application, you can't hardcode any text in any language as a part of your test case. To generalize this, it's recommended that you should add strings inside the `res/values/strings.xml` file.

Let's see how the internationalization test case is implemented using the following simple example.

An application consists of the **Connect** toggle button, which when clicked on, toggles itself to the **Disconnect** button. Below it, there is a TextView that displays all the connection logs generated by the application. The UI looks like the following screenshot:



Once the Connect button is clicked on, **Successful** is displayed in the Textview below it.

Instead of adding any hardcoded values for the text in the toggle button, we can internationalize it to use values specified in the `res/values/string.xml` file as follows:

```
<ToggleButton
    android:id="@+id/toggleConnection"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="50dp"
    android:textOn="@string/on"
    android:checked="true"
    android:textOff="@string/off"/>

<TextView
    android:id="@+id/tvConnectionLogs"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:maxLines="5"
    android:text="@string/connection_logs"
    android:layout_marginTop="120dip"/>
```

The values specified in the `string.xml` file are as follows:

```
<string name="on">Connect</string>
<string name="off">Disconnect</string>
<string name="connection_logs">Successful</string>
```

The following is the code of the test project `TestInterApp`, which has a test case named `testInterAppBlackBox` that clicks on the **Connect** button and toggles it to the **Disconnect** button. It then searches for the **Successful** text in the connection-log Textview present below the toggle button.

```
package com.android.testinterapp;

import android.app.Activity;
import android.test.ActivityInstrumentationTestCase2;

import com.android.interapp.MainActivity;
import com.jayway.android.robotium.solo.Solo;

public class TestInterApp extends
    ActivityInstrumentationTestCase2<MainActivity> {
```

```
private Solo solo;

public TestInterApp() {
    super(MainActivity.class);
}

@Override
protected void setUp() throws Exception {
    super.setUp();
    this.solo = new Solo(this.getInstrumentation(),
        this.getActivity());
}

public void testInterAppBlackBox() {

    // Gets the activity object from solo class using
    // method getCurrentActivity()
    Activity activity = solo.getCurrentActivity();

    // Gets the resource ID using the resource name and
    // type passed
    // in the getIdentifier method
    int connectOnId = activity.getResources().getIdentifier(
        "on", "string", activity.getPackageName());

    // Gets the string by using the resource id in the earlier
    // step
    String connect = activity.getResources().
        getString(connectOnId);

    this.solo.clickOnToggleButton(connect);

    // Similarly for the text view field, get the resource ID using
    // the resource name and type passed in the
    // getIdentifier method
    int connectionLogId = activity.getResources().
        getIdentifier("connection_logs", "string",
            activity.getPackageName());

    // Gets the string by using the resource id in the
    // earlier step
```

```

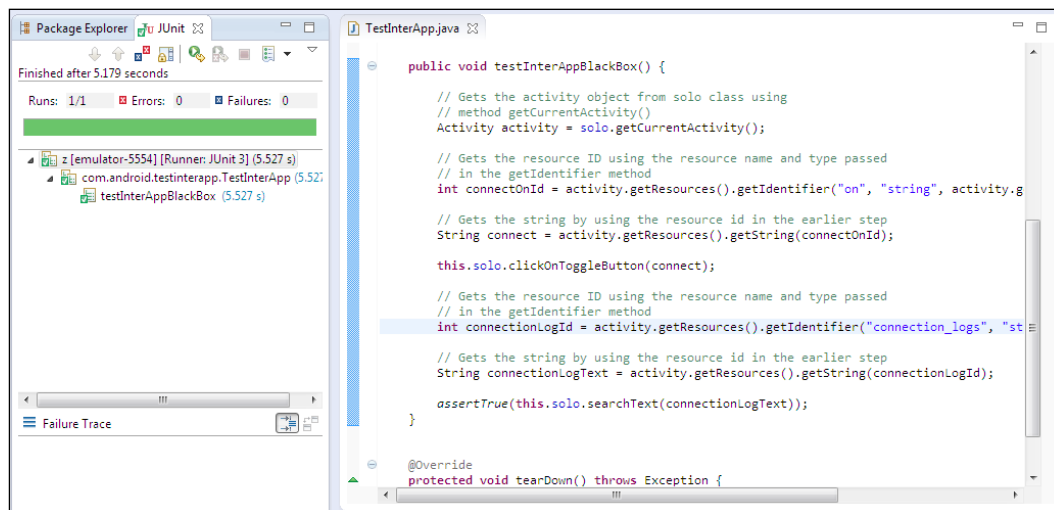
        String connectionLogText = activity.getResources().
            getString(connectionLogId);

        assertTrue(this.solo.searchText(connectionLogText));
    }

    @Override
    protected void tearDown() throws Exception {
        this.solo.finishOpenedActivities();
    }
}

```

The following screenshot shows the test case result in the JUnit console:



Test case and result

As a result of internationalization, if any user wants to change the language region-wise, there is no need to modify the Robotium test case. Instead, just change the variable value in the region language inside the string.xml file and the test case will work the same for all the regions and languages.

Summary

In this chapter, we have seen most of the Robotium API calls with their descriptions and how to use internationalization in Robotium. In the next chapter, you will learn how to access different web elements of the web view in an Android application using Robotium.

4

Web Support in Robotium

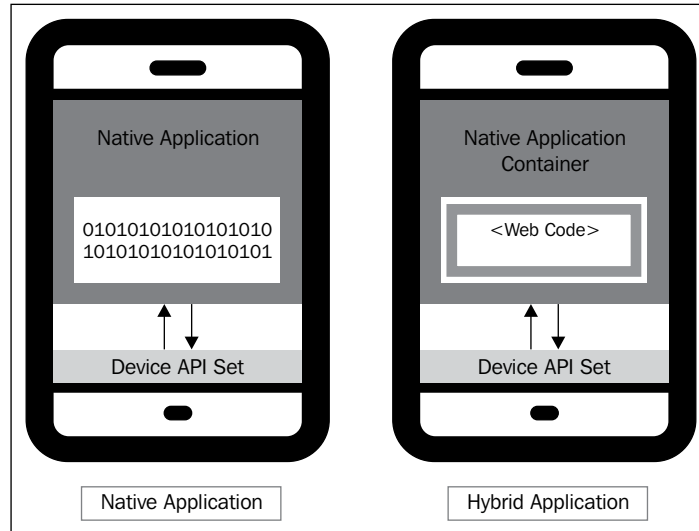
This chapter will brief you about accessing WebElements in Android using web support in Robotium. We will see these methods in the initial part of this chapter, and will move on to one simple example of testing a hybrid application.

API set

Web support has been added to the Robotium framework since Robotium 4.0 released. Robotium has full support for hybrid applications. There are some key differences between native and hybrid applications. Let's go through them one by one, as follows:

Native Application	Hybrid Application
Platform dependent	Cross platform
Run on the device's internal software and hardware	Built using HTML5 and JavaScript and wrapped inside a thin native container that provides access to native platform features
Need more developers to build apps on different platforms and learning time is more	Save development cost and time
Excellent performance	Less performance

The native and hybrid applications are shown as follows:



Let's see some of the existing methods in Robotium that support access to web content. They are as follows:

- `searchText (String text)`
- `scrollUp/Down()`
- `clickOnText (String text)`
- `takeScreenshot()`
- `waitForText (String text)`

In the methods specifically added for web support, the class `By` is used as a parameter. It is an abstract class used as a conjunction with the web methods. These methods are used to select different `WebElements` by their properties, such as ID and name.

The element used in a web view is referred to as a `WebElement`. It is similar to the `WebDriver` implemented in Selenium. The following table lists all the methods inside the class `By`:

Method	Description
<code>className (String className)</code>	Select a WebElement by its class name
<code>cssSelector (String selectors)</code>	Select a WebElement by its CSS selector
<code>getValue()</code>	Return the value
<code>id (String id)</code>	Select a WebElement by its id
<code>name (String name)</code>	Select a WebElement by its name
<code>tagName (String tagName)</code>	Select a WebElement by its tag name
<code>textContent (String textContent)</code>	Select a WebElement by its text content
<code>xpath (String xpath)</code>	Select a WebElement by its xpath

Some of the important methods in the Robotium framework, that aim at direct communication with web content in Android applications, are listed as follows:

- `clickOnWebElement (By by)`: It clicks on the WebElement matching the specified By class object.
- `waitForWebElement (By by)`: It waits for the WebElement matching the specified By class object.
- `getWebElement (By by, int index)`: It returns a WebElement matching the specified By class object and index.
- `enterTextInWebElement (By by, String text)`: It enters the text in a WebElement matching the specified By class object.
- `typeTextInWebElement (By by)`: It types the text in a WebElement matching the specified By class object. In this method, the program actually types the text letter by letter using the keyboard, whereas `enterTextInWebElement` directly enters the text in the particular.
- `clearTextInWebElement (By by)`: It clears the text in a WebElement matching the specified By class object.
- `getCurrentWebElements (By by)`: It returns the ArrayList of WebElements displayed in the active web view matching the specified By class object.

Before actually looking into the hybrid test example, let's gain more information about WebViews.

You can get an instance of `WebView` using the `Solo` class as follows:

```
WebView wb = solo.getCurrentViews(WebView.class).get(0);
```

Now that you have control of `WebView`, you can inject your JavaScript code as follows:

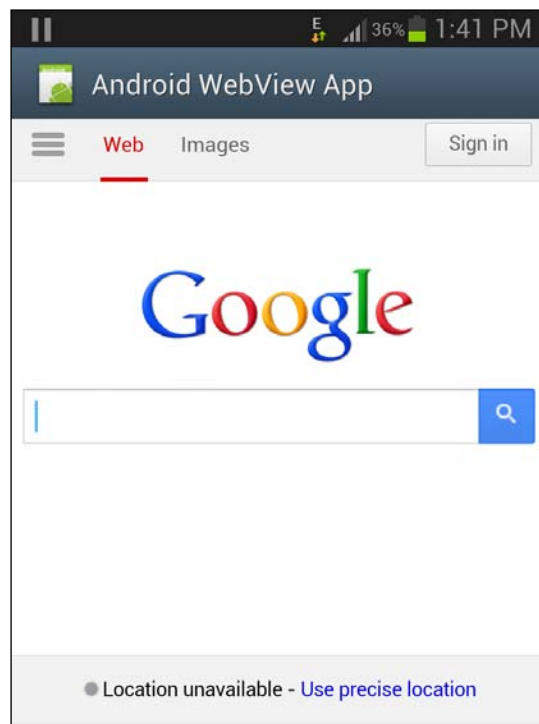
```
Wb.loadUrl("<JavaScript>");
```

This is very powerful, as we can call every function on the current page; thus, it helps automation.

Hybrid test example

Let's see one hybrid application, that is, the application under test, and create a simple test project to test this application.

The application provides a `WebView` control in its layout and loads Google's homepage in it (as shown in the following screenshot). You can see the source code of the application before moving onto writing test cases:



The source code of the `WebViewActivity.java` file is as follows:

```
public class WebViewActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.web_main);

        WebView webView = (WebView) findViewById(R.id.mainWebView);

        WebSettings webSettings = webView.getSettings();
        webSettings.setJavaScriptEnabled(true);

        webView.setWebViewClient(new CustomWebViewClient());
        webView.setScrollBarStyle(View.SCROLLBARS_INSIDE_OVERLAY);

        webView.loadUrl("http://www.google.co.in");
    }

    private class CustomWebViewClient extends WebViewClient {
        @Override
        public boolean shouldOverrideUrlLoading(WebView view,
            String url) {
            view.loadUrl(url);
            return true;
        }
    }
}
```

Add the following code in your `web_main.xml` layout file:

```
<WebView android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/mainWebView">
</WebView>
```

If you have not specified any permission in `AndroidManifest.xml`, provide the following permission:

```
<uses-permission android:name="android.permission.INTERNET" />
```

This sets up your application with `webView`. Now, let's write a test case that accesses some of the `WebElements` of the Google homepage, `webView`, and provides results.

Use the following code as a Google search test case in your hybrid test project:

```
// A test that searches for Robotium and asserts
// that Robotium is found.

public void testSearchRobotium()
{

    // Since Google's search form input box statement utilizes
    // a "q" in the name="q" parameter
    final By inputSearch = By.name("q");

    // Google search button utilizes "tsbb" in
    // the id="tsbb" parameter
    final By buttonSearch = By.id("tsbb");

    // Wait for a WebElement without scrolling.
    this.solo.waitForWebElement(inputSearch);

    // Types Robotium in the search input field.
    this.solo.typeTextInWebElement(inputSearch, "Robotium");

    //Assert that Robotium is entered in the input field.
    assertTrue("Robotium has not been typed",
        solo.getWebElement(inputSearch, 0).getText()
            .contains("Robotium"));

    // Clicks on the search button
    this.solo.clickOnWebElement(buttonSearch);

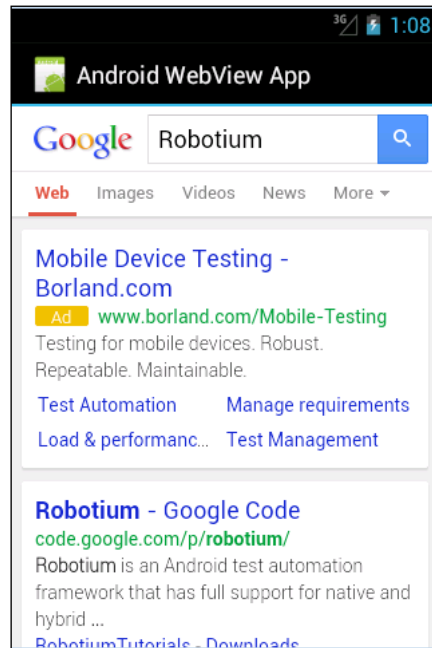
    // Waits for the results page for Robotium
    solo.waitForText("Results");

    // Takes the screenshot of the current active screen
    solo.takeScreenshot();

}
```

The preceding code types the text `Robotium` in the Google search box and clicks on the search button. It asserts if the word `Robotium` is not found in the input search bar. If it is present, the program clicks on the search button and the results page is displayed.

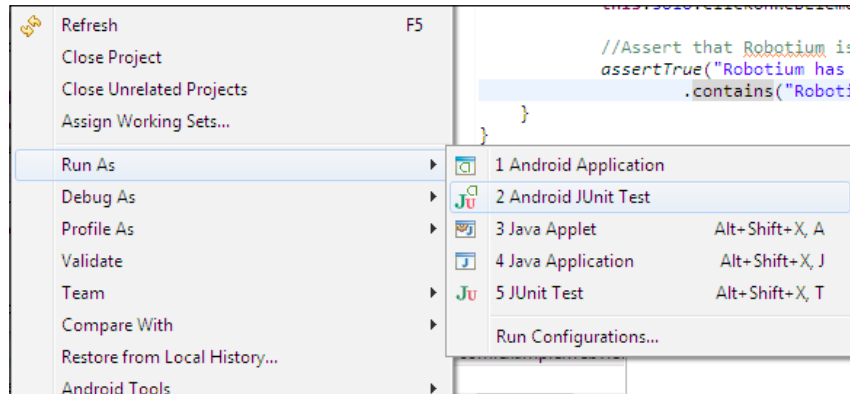
It then waits for the test results and takes a screenshot as follows:



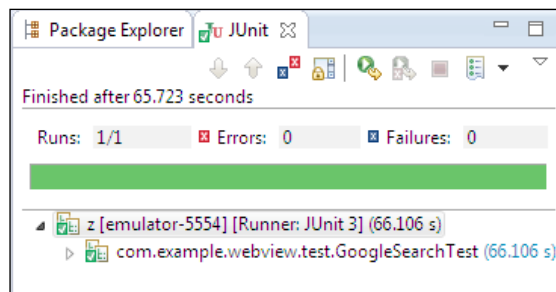
This screenshot is saved in the `/sdcard/Robotium-Screenshots/` directory by the API. It requires write permission (`android.permission.WRITE_EXTERNAL_STORAGE`) in the `AndroidManifest.xml` file of the application under test.

This result can be viewed under JUnit view. This view is automatically started when the test project is executed as an Android JUnit test. You can see the following screenshot to follow the process to run the test project.

Right-click on the test project, click on **Run As** and then on **2 Android JUnit Test**:



The testSearchRobotium test case is passed and indicated by the green bar, as shown in the following screenshot. It took around 66.1062 seconds to complete the test:



Summary

In this chapter, we learned about testing a hybrid application and different APIs that are used to access WebElements. With Robotium's web view support, we can basically test mobile webpages. In this way, we emulate the same conditions like opening mobile webpages using native browsers, because a native browser has tabs which contain WebView.

In the next chapter, we will compare the Robotium framework with other testing frameworks and see some interesting facts.

5

Comparison with Other Frameworks

This chapter provides a comparison between Robotium and other testing frameworks based on certain parameters. This will provide you a way to select a proper framework according to your project needs. In this chapter, we will compare Robotium with MonkeyRunner, Robolectric, UI Automator, and Calabash frameworks.

MonkeyRunner

MonkeyRunner is a tool used to write programs that can access the Android emulator/device from outside of the Android code. Python programs are written to install the Android test app and send keystrokes to the app. The program takes the screenshots of the Android UI and sends it to the workstation for storage.

MonkeyRunner is an API, not a program. It uses **Jython** (an implementation of python) that uses the Java programming language.

Since MonkeyRunner is a module of Python, you can do anything that is supported by Python. All you need to do is create a Python program and add MonkeyRunner to it and you're done!

Let's see the difference between Robotium and MonkeyRunner in the following table:

Features	Robotium	MonkeyRunner
Object selection	Object selection is based on attributes such as index, text/ name, image, and ID.	Object selection is based on its location (x, y co-ordinates), which can change when the application evolves. There are high chances that touch events can't be used as the exact location is not provided.
Actions	It can perform actions only on the tested app.	It can click throughout the device, that is, all the applications present.
Assertion	JUnit based. A red/ green bar is displayed for asserts (verification).	Screenshot-based verification.
Language	Java.	Python scripts.
Installation	The Robotium JAR can be imported inside the Eclipse plugin and the test case can be executed as a . apk file.	To use MonkeyRunner, run the monkeyrunner tool present in <android sdk>/tools/ and pass the filename to be used as a test case. It does not install any program inside the emulator/ device.

There are some points common to the two frameworks. They can run on an emulator/ device and control devices/ emulators from a workstation by sending specific commands and events from an API.

In the Android testing domain, different frameworks are present for different needs. Since Robotium is mostly used for UI testing, it doesn't support some of the following features of MonkeyRunner:

- Extensible automation
- Multiple application and device control

Robolectric

Robolectric is a test framework that mocks a part of the Android framework and allows the running of test cases directly on the **Java Virtual Machine (JVM)** with the help of the JUnit 4 framework. The most important thing about Robolectric is that it does not need an emulator/device.



Robolectric contains the shallow Android objects that behave like objects present in the Android SDK.

Let's see the difference between Robotium and Robolectric in the following table:

Features	Robotium	Robolectric
Emulator/Device	Robotium needs either an emulator or a device to execute tests.	Robolectric does not need any emulator/device to execute tests. This is why it is much faster than Robotium.
Build server	It needs an emulator or a device on the build server to run test cases; otherwise, the test project can't be added to the build process.	It can be configured easily on the build server.
Test-driven development	It is used to test on an actual Android device and the API edges that are not simulated by Robolectric.	It helps to speed up the test driven development cycle more than Robotium.
Instrumentation	It uses JUnit 3 instrumentation testing.	It uses JUnit 4 non instrumentation testing.

UI Automator

UI Automator is a Java library used to create customized-UI test cases for an android application and it provides an execution engine to automate and run test cases.

Let's see the difference between Robotium and UI Automator in the following table:

Features	Robotium	UI Automator
Cross-application package	Robotium can't cross the application package boundary.	UI Automator can cross application package boundaries. For example, if your application opens the gallery and clicks on any album, this can be achieved using UI Automator. The gallery is another application package and clicking on an album inside the gallery is a cross-application operation.
API set	Robotium has an enormous API set, containing methods for clicking on views, getting views, and so on. Due to this, Robotium provides more control over testing than UI Automator.	UI Automator contains methods to click and get views, but implements a different access to these views.
API-level support	Robotium supports API level 4 and above.	UI Automator supports devices with only API level 16 (or above) and doesn't go back to support older API levels; thus, there is no backward compatibility.
Integration with IDE	Robotium integrates smoothly with Eclipse IDE.	UI Automator integration with IDE is cumbersome as you need to manually add the JUnit library with Android.jar and uiautomator.jar, and build it using Ant.
Web support	Robotium has full support for Web Elements in the application.	UI Automator lacks this feature.

Calabash

Calabash is the cross-platform that enables you to write automated-functional acceptance tests for mobile applications, supporting Android and iOS Native apps.

Let's see the difference between Robotium and Calabash in the following table:

Features	Robotium	Calabash
Language	Robotium test cases are written in Java	You don't need to write your tests in Java, you can use a more flexible language Ruby , which fits better
Control	Robotium tests need to be controlled from a device	Calabash tests can be controlled from a computer instead of a device
Rotation	Robotium can set the device orientation to Landscape or Portrait	Calabash-Android doesn't mimic the phone rotation to Landscape or Portrait

So, which is better? Robotium or Calabash? I think both are good. They are still being improved and more versions are being released.

You can always use the `android.test.ActivityInstrumentationTestCase2` class when neither satisfies your needs.

As discussed, each framework has its pros and cons and is available for different needs of the automation testers. As we have seen from the previous comparison, Robotium and Calabash are quite popular and are leading all the way in automated UI testing.

Summary

In this chapter, we compared Robotium with different testing frameworks based on different factors and came to the conclusion that all the frameworks are used according to different needs. No framework is perfect, there are always some pros and cons associated with it.

In the next chapter, we will look into the remote control feature and its usage in Robotium.

6

Remote Control in Robotium

This chapter will introduce you to the Software Automation Framework Support and the working of the Remote Control in Android. It helps to test the connection of an Android device to a remote client via the TCP Messenger Service. By the end of this chapter, you will learn the internal mechanism of how the Remote Control is used with a test APK that does not have Internet permissions and connects to a remote client with the help of the Android socket service.

Software Automation Framework Support

The main function of the **Software Automation Framework Support (SAFS)** is to fully integrate the use of Robotium for Android testing with a large set of users that use different technologies using the SAFS.

In Android, sockets, UDP and RMI are not supported. Thus, the test application is not supposed to have access to the TCP sockets if AUT doesn't have the following permissions:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

The reason a Remote Control is introduced is that there is a generic way that uses the socket service in Android, which is completely independent of the test application and AUT.

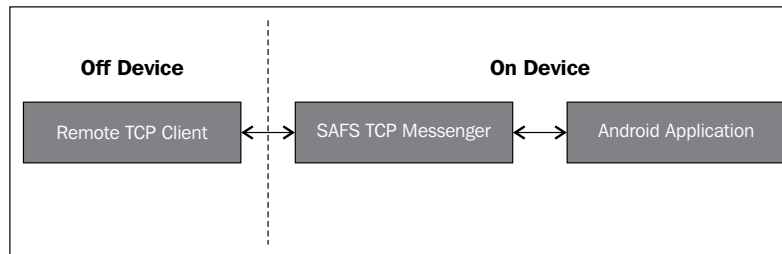
Let's see how a Remote Control using SAFS works.

Working of a Remote Control for Android



The working of a Remote Control for Android can be defined by the following steps:

1. The user installs a socket service on the device/emulator. This socket service has full Internet permission. The TCP message exchange protocol is used by the remote process to send/receive the commands and share the results with the already installed socket service.
2. The test application to be written for the AUT consists of the code to bind to this socket service. This test app doesn't need any Internet permission. It can rely on the socket service for the data results.



The TCP Messenger Service (socket service) acts as the mediator between the remote test controller (Windows, Linux, Mac, and so on) that communicates over the TCP sockets and an Android test package. It expects two-way communication for receiving commands and returning test data/results. Due to this service, a test package with no Internet permissions can be controlled via TCP sockets.

3. The test application receives the commands from the socket service, performs the tests and returns the data, status needed by the test controller.

The application to be tested should not be hardcoded in the test application. This can be easily automated without the developer/tester code by updating the `android:targetPackage` attribute in the `AndroidManifest.xml` file. Thus, at the pre-test runtime, information out of the target APK can be extracted and can repackage an otherwise unmodified test application APK with an updated `AndroidManifest.xml` file for the target package. During runtime, the test APK can identify the target package and the initial activity to be launched through the `PackageManager`, `PackageInfo`, and `ApplicationInfo` calls.

Robotium Remote Control helps test cases to be executed from JVM, which enables Robotium to work with JVM frameworks such as JUnit 4. For the Remote Control, another `Solo` class is present at `com.jayway.android.robotium.remotecontrol.solo`. It provides the Remote Control API to the embedded traditional Robotium `Solo` class.

Since the Remote Control `Solo` class runs outside the device/emulator, that is, in the context of the JUnit test, it can't access the **Application Under Test (AUT)** objects. It does not abort or stop on a test failure. You can see the different methods and information present for it at the following source:

<http://safsdev.sourceforge.net/doc/com/jayway/android/robotium/remotecontrol/solo/Solo.html>

It uses the TCP Messenger Service for Android. You can download Robotium Remote Control release for Android testing with Robotium from the following URL:

<http://sourceforge.net/projects/safsdev/files/RobotiumRemoteControl/>

You can get the latest SAFS download from the following URL:

<http://sourceforge.net/projects/safsdev/files/latest/download>

Once you download Robotium Remote Control, you will find the `SoloRemoteControl` project already installed with it. It should be used as a reference for your own Java development project. The `src` folder inside the `SoloRemoteControl` project contains all the source code in the `robotium-remotecontrol.jar` file.

Robotium Remote Control using SAFS

SAFS tests are not wrapped up as JUnit tests and the SAFS Remote Control of Robotium uses an implementation that is NOT JUnit based. Also, there is no technical requirement for a JUnit on the Remote-Control side of the test.

The test setup and deployment of the automation of the target app can be achieved using the SDK tools. These tools are used as part of the test runtime such as adb and aapt. The existing packaging tools can be used to repackage a compiled Robotium test with an alternate `AndroidManifest.xml` file, which can change the target application at runtime.

SAFS is a general-purpose, data-driven framework. The only thing that should be provided by the user is the target package name or APK path arguments. The test will extract and redeploy the modified packages automatically and then launch the actual test.

Traditional JUnit/Robotium users might not have, or see the need for, this general-purpose nature, but that is likely because it was necessary for the previous Android tests to be JUnit tests. It is required for the test to target one specific application. The Remote Control application is application specific. That's why the test app with the Remote Control installed in the device no longer needs to be an application.

The Remote Control in Robotium means there are two test applications to build for any given test. They are as follows:

- Traditional on-device Robotium/JUnit test app
- Remote Control app

These two build projects have entirely different dependencies and build scripts.

The on-device test app has the traditional Robotium/Android/JUnit dependencies and build scripts, while the Remote Control app only has dependencies on the TCP sockets for communications and Robotium Remote Control API.

The implementation for the remote-controlled Robotium can be done in the following two pieces:

- **On Device:** `ActivityInstrumentationTestCase2.setup()` is initialized when Robotium's `Solo` class object is to be used for the **RobotiumTestRunner (RTR)**. The RTR has a Remote Control Listener and routes remote control calls and data to the appropriate `Solo` class methods and returns any results, as needed, to the Remote Controller. The on-device implementation may exploit test-result asserts if that is desirable.

- **Remote Controller:** The `RemoteSolo` API duplicates the traditional `Solo` API, but its implementation largely pushes the data through the Remote Control to the RTR, and then receives results from the Remote Controller. The Remote Control implementation may exploit any number of options for asserting, handling, or otherwise reporting or tracking the test results for each call.

As you can see, the Remote-Control side only requires a `RemoteSolo` API without any specific JUnit context. It can be wrapped in a JUnit context if the tester desires it, but it is not necessary to be in a JUnit context.

The sample code and installation of Robotium Remote Control can be accessed in the following link:

<http://code.google.com/p/robotium/wiki/RemoteControl>

Summary

In this chapter, you learnt about the SAFS framework and its usage in Robotium to implement Robotium Remote Control. In the next chapter, you will be introduced to the Robotium utilities.

7

Other Robotium Utilities

This chapter consists of various utilities present in the Robotium Framework. These utilities include the `RobotiumUtils` class, XPath usage, Robotium usage for the already installed Android applications, and the signature process involved during the application sign-unsigned operation to perform tests. Let's see these utilities one by one in this chapter.

The RobotiumUtils class

The `RobotiumUtils` class was introduced in the Robotium v4.2. It contains the utility methods. Let's see the API set for the class `RobotiumUtils`.



API set

The `RobotiumUtils` class provides methods for sorting, ordering, filtering views, and so on. The different methods for providing these features are as follows:

- The following method filters the views based on the class passed as a parameter:

```
RobotiumUtils.filterViews(Class classToFilterBy,  
    Iterable viewList)
```
- The following method filters all the views that are not present in the `classSet` parameter:

```
RobotiumUtils.filterViewsToSet(Class<android.view.View>[]  
    classSet, Iterable<android.view.View> viewList)
```
- The following method checks if a view matches the `stringToMatch` variable and returns the number of matches found:

```
RobotiumUtils.getNumberOfMatches(String stringToMatch,  
    TextView targetTextView, Set<TextView> textviewList)
```
- The following method removes all the views whose visibility is invisible:

```
RobotiumUtils.removeInvisibleViews(Iterable viewList)
```
- The following method sorts all the views by their location on the screen:

```
RobotiumUtils.sortViewsByLocationOnScreen(List viewList)
```
- The following method filters the views collection and returns the views that match a specified regular expression.

```
RobotiumUtils.filterViewsByText(Iterable viewList,  
    string regex)
```



You can get more information about these methods from the following link:

<http://robotium.googlecode.com/svn/doc/com/jayway/android/robotium/solo/RobotiumUtils.html>

XPath API and syntax

Web Elements can be located by using an XPath-style expression. This helps testers to find the elements of DOM while keeping the tests resistant to the application UI. Note that this applies only to Web Elements inside the WebView application.



XPath is basically used to navigate through the attributes and elements in an XML document. A simple path expression can be used to navigate through XML documents using XPath.

Let us see a simple example that provides a basic understanding of how XPath is to be used:

The following is the XML file that we have to parse using XPath:

```
<?xml version="1.0"?>
<catalog>
  <book id="z24">
    <author>Hrushikesh</author>
    <title>Robotium Automated Testing For Android</title>
    <genre>Computer</genre>
    <description>Efficiently automate test cases for Android
      applications using Robotium </description>
  </book>

  <book id="az24">
    <author>Bharati</author>
    <title>Mathematics MHTCET</title>
    <genre>Objective</genre>
    <description>A comprehensive guide to Mathematics for
      MHTCET</description>
  </book>
</catalog>
```

Suppose you want to extract the names of the authors from the previous XML file. The function to do it using XPath is as follows:

```
private void parseDataUsingXPath() {
    // create an InputSource object from /res/xml
    InputSource inputSrc = new InputSource(
        getResources().openRawResource(R.xml.data));

    // Instantiate the XPath Parser instance
    XPath xPath = XPathFactory.newInstance().newXPath();

    // Create the xpath expression
    String xPathExpression = "//author";

    // list of nodes queried
    NodeList nodes = (NodeList)xpath.evaluate(expression,
        inputSrc, XPathConstants.NODESET);

    // if author node found, then add to authorList i.e. the list
    // of authors instantiated in the activity where this function
    // is used
    if(nodes != null && nodes.getLength() > 0) {
        int len = nodes.getLength();
        for(int i = 0; i < len; ++i) {
            Node node = nodes.item(i);
            authorList.add(node.getTextContent());
        }
    }
}
```

You can find Web Elements by using the following XPath expression example:

```
solo.getWebElement(By.xpath("//*[@id='nameOfelementID']/tbody/
tr[11]/td[2]/input[1]"), 0);
where By.xpath(<path expression>, index);
```

Robotium for pre-installed applications

Robotium allows writing and running test cases for the pre-installed applications in Android. To do this, you need to perform the re-sign operation on the target application with the debug key that you are using for the test project. This requires access to the /system/app folder on the device. To access this folder, you must have a rooted device.



Please note that some of the preinstalled applications for which the re-sign operation is performed don't work properly. These applications don't even show up when re-signed with the new debug key.

To re-sign an APK file, you can follow the listed steps:

1. Log in as root: `adb root`.
2. Remount using the following code


```
adb remount
adb pull /system/app/ApplicationName.apk
```
3. Re-sign the `ApplicationName.apk` file, so that it has the same certificate debug key signature signing as the `adb pull /data/system/packages.xml` test project.
4. Open `packages.xml` and remove the following code:


```
<package name="com.ApplicationName">
.....
</package>
```
5. Push `packages.xml` back to the `adb push packages.xml /data/system` device.
6. Restart your device.
7. Push the re-signed `ApplicationName.apk` file back to the `adb push ApplicationName.apk /system/app` device.

Test for only APK

The APK file to be tested should have the same certificate signature as the test project. If the signatures are different, a mismatch will occur.



- If you know the APK certificate signature, sign the test project with the same signature.
- If the APK is not signed, sign the APK with the Android debug key signature.
- If you don't know the APK certificate signature, delete its certificate signature and provide the debug key signature to the APK and your test project.

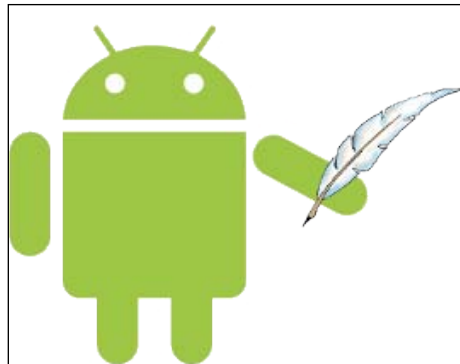
Signature process

In Android OS, all the applications are signed with the certificate whose private key is held by the app developer.

There are two build modes to build your Android application. They are as follows:

- Debug mode
- Release mode

The Android build process signs the application depending on the type of build mode. The Debug mode is mostly used when you are developing and testing the application, whereas the Release mode is used when you want to distribute the released version of your application to the users, or publish the application in the marketplace such as the **Google Play Store**.



The `Keytool` utility is used to create a debug key when you are using a build mode. The alias and password of the debug key are known to the SDK build tools. This is why tools don't prompt you for the debug key's alias and password every time the program is compiled.

The private key is used when you build your application in the Release mode. If you don't have a private key with you, the `Keytool` utility creates one for you. When your application is compiled in the Release mode, the build tools use your private key, along with the `jarsigner` utility to sign the application APK file.

Whenever you debug or run your application using Eclipse with the ADT plugin installed, the debug key signing process happens automatically in the background. The same signing process is followed when using an Ant script with the debug option enabled to build the application. The *Eclipse Export wizard* helps automating the release signing process or modifying the Ant build script and building applications with the release option.

To unsign a signed APK and then sign it with the debug key, perform the following steps:

1. Unzip the APK file using any ZIP extractor.
2. Delete the `Meta-INF` folder.
3. Rezip the extracted files to an APK file and change the extension from `appname.apk.zip` to `appname.apk`. This way, you can un-sign the signed APK!
4. To sign this APK with the Android debug signature key, run the following command in the command prompt:

```
> jarsigner -keystore ~/.android/debug.keystore -storepass  
    android -keypass android appname.apk androiddebugkey  
  
> zipalign 4 appname.apk tempappname.apk
```


The `jarsigner` utility can be found under the JDK binaries and `zipalign` is an Android SDK part.

The `zipalign` tool is used for optimization of the Android APK files. Its basic purpose is to have the uncompressed data alignment relative to the start of the file.

With the help of Eclipse, you can also sign and export your Android app. Perform the following steps:

1. Right-click on the Android project and navigate to **Android Tools | Export Signed Application Package**.
2. In the **Export** Android application wizard, select the project you want to export, and then click on **Next**.
3. The **Keystore Selection** screen will appear.
4. If you don't have the existing keystore, create a new keystore. You should be able to create a keystore by entering the location and password.
5. After navigating to the folder you want to use, type a name in the **File name:** field in the file browse window, for example, `hrushikesh.keystore`. Then, you should be able to proceed with the key creation.

For more information on APK signing, please refer to the following link:

<http://developer.android.com/tools/publishing/app-signing.html>

Summary

In this chapter, you learned about the different utilities present in the Robotium Framework and their usage. In the next chapter, we will see the integration mechanism of Robotium with Maven, with some examples.

8

Robotium with Maven

This chapter introduces the use of the Maven tool to attach an Android project for adding to a build process. This chapter also explains different configurations/installations you need to use Robotium with Maven.

Automate Android app built with Maven



Maven is basically a software project management tool based on the concept of **Project Object Model (POM)**, which is required at the project root and helps to build a project.

It can manage a project's build, reporting, and documentation from a central piece of information.

Maven features

Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. To attain this goal, there are several areas of concern that Maven attempts to deal with:

Feature	Description
Repeatable Builds	You can build the project repeatedly on the build server.
Focus on Automation	Maven puts you in the right mindset of automating processes in software development.
Dependency Management	Maven will resolve and manage your dependencies.
Standardization	New developers, who understand Maven, will instantly know how to build, release, test, and thus, remove a lot of learning overhead.
Plugins	There are a lot of plugins available to carry out different tasks. These are configured by adding a reference into the pom.xml file.
Testing	This gives you the ability to run tests and integration tests as part of your project lifecycle.

To get Maven up and running for Android, you must use the **Android Maven Plugin** for the existing Eclipse project. You can install Maven from the link mentioned in the preceding figure.

The homepage of the Android Maven Plugin is located at the following website:

<https://code.google.com/p/maven-android-plugin/>

Setting up Android SDK and ADT

You can create/build an Android project using the command-line tool that is provided by Android SDK tools. The same functionality is provided by ADT for Eclipse. You can also export Android applications manually via the Eclipse export wizard. Currently, Apache Ant is mostly used by the tools provided by Android SDK to build and deploy an application.



The new Gradle-based build system for Android apps is a huge improvement over the older Eclipse-, Ant-, and Maven-based applications. It has a simple declarative syntax and makes building different variants of your app (for example, staging vs. production) very easy. Gradle is also the default build system for the new Android Studio IDE, so there are lots of good reasons to migrate your apps.

Android Studio will be based on Gradle. In Android Studio, there won't be ADT as it will have built-in Android utilities. Gradle internally uses Maven repositories to manage the dependencies, which ultimately help to make the support of Maven quite easy.

Setting up the environment PATH for Android Tools

When you want to build an Android project outside Eclipse, you mostly need to use a command line or shell. To achieve this, you need to make sure that the `tools` and `platform-tools` folders of the Android SDK are part of the PATH environment variable. To do this, please follow the steps:

1. Set the environment variable `ANDROID_HOME` to the path of your Android SDK.

Windows users:

- From the desktop, right-click on **My Computer** and click on **Properties**.
- Click on the **Advanced System Settings** link in the left column.
- In the **System Properties** window, click on the **Environment Variables** button and add new variable with name `ANDROID_HOME`

Unix-based system users:

- Export `ANDROID_HOME=/path/to/android/sdk`
2. Also, add `$ANDROID_HOME/tools` and `$ANDROID_HOME/platform-tools` to `$PATH` (for Windows, `%ANDROID_HOME%\tools` and `%ANDROID_HOME%\platform-tools`).
 3. All MacOS users, please note that for the path to work on the command line and in IDEs started by `launchd`, you need to set it in `/etc/launchd.conf`.

Build Android tests using Maven

Create an Android project named `com.android.build.maven`. In this section, let us concentrate on creating a build for your Android test application using Maven. Add the correct `pom.xml` file, as shown in the following project directory and provide commands to Maven to build, install, and run your application.

The following is the code for the `pom.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
apache.org/maven-v4_0_0.xsd">
...
...
...
</project>
```

You can refer to the full source code of the `pom.xml` file in `chapter8_code1`.

The preceding `pom.xml` file is very similar to the `pom.xml` file in the main project, but this has several dependencies.

The dependency for `apk` is to enable the Android Maven plugin to find `apk` that it will run the tests against on the device/emulator. The dependency to the JAR file is to enable the compiler to find your Java classes from the main project. To achieve this, you use the provided scope so the classes are not actually included into your test project.

The Android Maven plugin that is provided under the build section in `pom.xml` will now run tests automatically on `mvn install` using instrumentation, just like Eclipse does. It uses the same underlying tools.

When you have only a single emulator/device connected, the automatic execution will work. If you have more than one device/emulator running, you will need to specify which device to use with either of these as command-line options:

- `-Dandroid.device=usb`
- `-Dandroid.device=emulator`
- `-Dandroid.device=specificdeviceid`

You can also disable instrumentation tests with this command-line option:

- `-Dandroid.enableIntegrationTest=false`

The default properties can be set in `pom.xml` as follows:

```
<project>
...
  <properties>
    <android.device>emulator</android.device>
  </properties>
...
</project>
```

Build your app with Maven and deploy it to the device/emulator using the following command:

```
mvn install android:deploy
```

Using Maven, you can also start and stop an Android virtual device automatically. The application can be started via Maven using the following command:

```
mvn3 android:run
```



We have to be in the directory of the project, where `pom.xml` is located.

Summary

In this chapter, you learnt different methods to use Maven with Android and Robotium tests. You also learnt different Maven commands to interact with Android applications, that is, the test application.

What do you say? Are you ready to create and automate test cases for your Android project? I bet you are! Go ahead, enjoy automated testing for Android using Robotium and don't forget to tell the community about it!

Join the forums at <https://groups.google.com/forum/#!forum/robotium-developers> and be a part of this amazing Robotium developer's community. We're waiting for you!

Thanks for reading the book and supporting open source technologies. Hope to see you around soon!

Index

A

ADT (Android Developer Tools)

- about 9
- installing 10-12

Android

- Remote Control, working 60, 61

Android app build

- ADT, setting up 74
- Android SDK, setting up 74
- automating, with Maven 73

Android build process 70

Android environment

- setting up, for Robotium 9

Android environment setup, for Robotium

- ADT, installing 10-12
- Android SDK, adding to Eclipse 12
- Android SDK, installing 9
- AVD, setting up 14, 15
- latest SDK version, installing 13
- requisites 9

AndroidManifest.xml file

- package name, adding in 32, 33

Android Maven Plugin 74

Android SDK

- adding, to Eclipse 12

Android tests

- building, Maven used 76, 77

Android tools

- environment PATH, setting up for 75

Android Virtual Device (AVD) 6

API calls 38

API set 45

API set, RobotiumUtils class 66

APK file

- signature process 70-72

- testing 70

APK signing

- URL 72

AUT (Application Under Test)

- about 17, 39
- creating 18-25

AUT objects 61

automated testing 5

C

Calabash

- about 57
- versus Robotium 57

D

Dalvik Virtual Machine (DVM) 8

E

Eclipse

- Android SDK, adding to 12

environment PATH

- setting up, for Android Tools 75

F

features, Maven

- about 74
- dependency management 74
- focus on automation 74
- plugins 74
- repeatable builds 74
- standardization 74
- testing 74

G

Google Play Store 70
Gradle 75
Graphical User Interface (GUI) 5

H

hybrid application
 versus native application 45
hybrid test example 48-52

I

installation, ADT 10-12
internationalization 40
internationalization test case
 implementing 40-43

J

jarsigner utility 71
Java Development Kit (JDK) 9
Java Virtual Machine (JVM) 55
Jython 53

K

Keytool utility 71

M

Maven
 about 73
 features 74
 used, for automating Android app build 73
 used, for building Android Tests 76, 77

MonkeyRunner
 about 53
 versus Robotium 54

N

native application
 versus hybrid application 45

P

package name
 adding, in AndroidManifest.xml file 32, 33
Project Object Model (POM) 73

R

Remote Control
 working, in Android 60, 61
Remote Control, in Robotium
 SAFS used 61, 63
remote-controlled Robotium
 implementing 62
RemoteSolo API 63
resource ID
 in Robotium 39
Robolectric
 about 55
 versus Robotium 55
Robotium
 about 5, 6
 benefits 8
 features 8
 for preinstalled applications 68, 69
 hybrid application versus native
 application 45
 resource ID 39
 test case code 33, 34
 versus Calabash 57
 versus MonkeyRunner 54
 versus Robolectric 55
 versus UI Automator 56
Robotium framework
 methods 47
 web support 45, 46
Robotium library
 adding 30-32
RobotiumTestRunner (RTR) 62
RobotiumUtils class
 about 65
 API set 66
Robotium v 4.2 65

S

SAFS (Software Automation Framework Support) 59, 62

signed APK

 unsigning 71

Solo class

 about 37

 constructors 37

T

test case

 creating 28, 30

 running 35, 36

test project

 creating, for ZCalculator application 26, 27

triangularization workflow diagram 7

U

UI Automator

 about 56

 versus Robotium 56

X

XPath API 66, 67

Z

ZCalculator application

 AUT, creating 17-25

 Robotium library, adding 30-32

 test case, creating 28, 30

 test case, running 35, 36

 test project, creating for 26, 27

zipalign tool 72



Thank you for buying **Robotium Automated Testing for Android**

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.



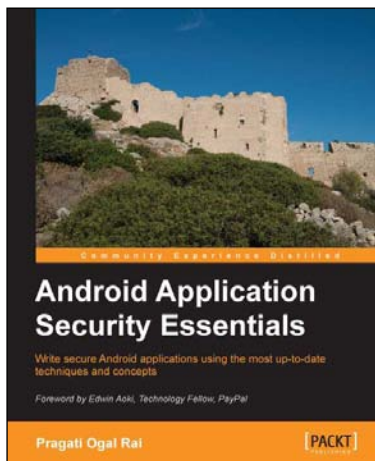
Instant Spring for Android Starter

ISBN: 978-1-78216-190-5

Paperback: 72 pages

Leverage Spring for Android to create RESTful and OAuth Android apps

1. Learn something new in an Instant!
A short, fast, focused guide delivering immediate results
2. Learn what Spring for Android adds to the Android developer toolkit
3. Learn how to debug your Android communication layer observing HTTP requests and responses



Android Application Security Essentials

ISBN: 978-1-84951-560-3

Paperback: 218 pages

Write secure Android applications using the most up-to-date techniques and concepts

1. Understand Android security from kernel to the application layer
2. Protect components using permissions
3. Safeguard user and corporate data from prying eyes
4. Understand the security implications of mobile payments, NFC, and more

Please check www.PacktPub.com for information on our titles



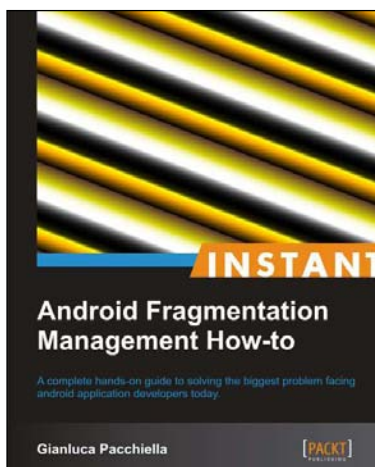
Android Development Tools for Eclipse

ISBN: 978-1-78216-110-3

Paperback: 144 pages

Learn how to set up, build, and publish Android projects quickly using Android Development Tools for Eclipse

1. Build Android applications using ADT for Eclipse
2. Generate Android application skeleton code using wizards
3. Advertise and monetize your applications



Instant Android Fragmentation Management How-to

ISBN: 978-1-78216-086-1

Paperback: 66 pages

A complete hands-on guide to solving the biggest problem facing Android application developers today

1. Learn something new in an Instant! A short, fast, focused guide delivering immediate results
2. Learn how to write apps that work on any Android version
3. Ready to use code to solve any compatibility issue
4. Get hands-on with the biggest issue that faces Android developers

Please check www.PacktPub.com for information on our titles