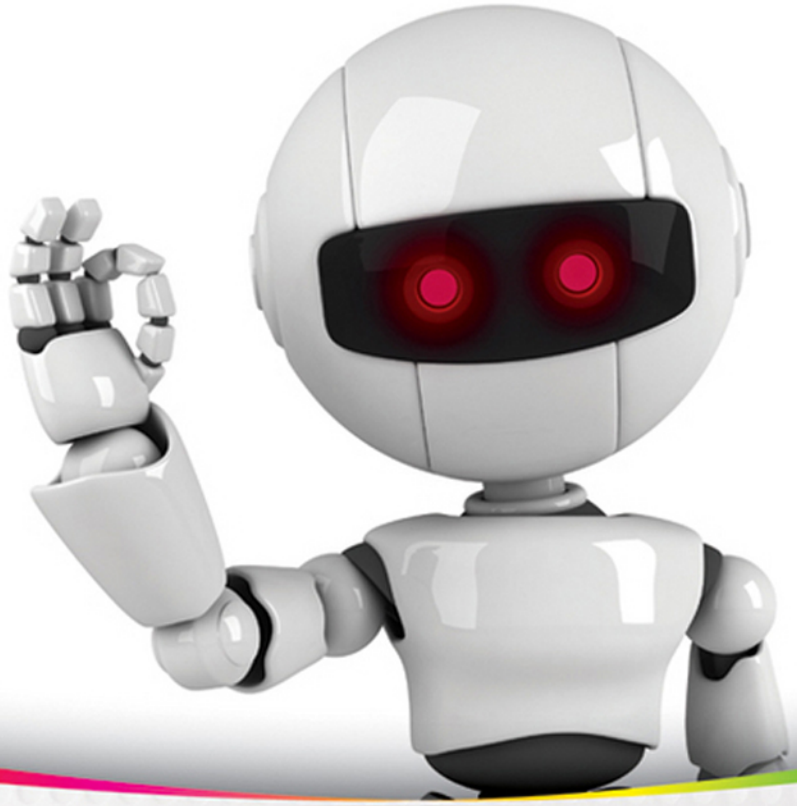
Join the discussion @ p2p.wrox.com

COVERS ANDROID 4



Beginning Android™ 4 Application Development

IN FULL COLOR

Wei-Meng Lee

BEGINNING ANDROID™ 4 APPLICATION DEVELOPMENT

BEGINNING

Android™ 4 Application Development

BEGINNING

Android™ 4 Application Development

Wei-Meng Lee

John Wiley & Sons, Inc.

Beginning Android™ 4 Application Development

Published by
John Wiley & Sons, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Published simultaneously in Canada

ISBN: 978-1-118-19954-1
ISBN: 978-1-118-22824-1 (ebk)
ISBN: 978-1-118-24067-0 (ebk)
ISBN: 978-1-118-26538-3 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Control Number: 2011945560

To my family:

Thanks for the understanding and support
while I worked on getting this book ready.
I love you all!

ABOUT THE AUTHOR

WEI-MENG LEE is a technologist and founder of Developer Learning Solutions (`www.learn2develop.net`), a technology company specializing in hands-on training on the latest mobile technologies. Wei-Meng has many years of training experience and his training courses place special emphasis on the learning-by-doing approach. This hands-on approach to learning programming makes understanding the subject much easier than reading books, tutorials, and other documentation.

Wei-Meng is also the author of Beginning iOS 5 Application Development (Wrox, 2010) and Beginning Android Application Development (Wrox, 2011). Contact Wei-Meng at `weimenglee@learn2develop.net`.

ABOUT THE TECHNICAL EDITOR

CHAIM KRAUSE is a Simulation Specialist at the US Army's Command and General Staff College where he develops various software products on a multitude of platforms, from iOS and Android devices to Windows desktops and Linux servers, among other duties. Python is his preferred language, but he is multilingual and also codes in Java and JavaScript/HTML5/CSS, and others. He was fortunate to begin his professional career in the software field at Borland where he was a Senior Developer Support Engineer for Delphi. Outside of computer geek stuff, Chaim enjoys techno and dubstep music and scootering with his two sled dogs, Dasher and Minnie.

CREDITS

EXECUTIVE EDITOR
Robert Elliott

SENIOR PROJECT EDITOR
Ami Sullivan

TECHNICAL EDITOR
Chaim Krause

PRODUCTION EDITOR
Kathleen Wisor

COPY EDITOR
Luann Rouff

EDITORIAL MANAGER
Mary Beth Wakefield

FREELANCER EDITORIAL MANAGER
Rosemarie Graham

ASSOCIATE DIRECTOR OF MARKETING
David Mayhew

MARKETING MANAGER
Ashley Zurcher

BUSINESS MANAGER
Amy Knies

PRODUCTION MANAGER
Tim Tate

VICE PRESIDENT AND EXECUTIVE GROUP PUBLISHER
Richard Swadley

VICE PRESIDENT AND EXECUTIVE PUBLISHER
Neil Edde

ASSOCIATE PUBLISHER
Jim Minatel

PROJECT COORDINATOR, COVER
Katie Crocker

PROOFREADER
Nancy Carassco

INDEXER
Johnna VanHoose Dinse

COVER DESIGNER
Ryan Sneed

COVER IMAGE

ACKNOWLEDGMENTS

WRITING THIS BOOK HAS been a roller-coaster ride. Working with just-released software is always a huge challenge. When I first started working on this book, the Android 4 SDK had just been released; and wading through the documentation was like finding a needle in a haystack. To add to the challenge, the Android emulator for the tablet is extremely slow and unstable, making the development process very laborious.

Now that the book is done, I hope your journey will not be as eventful as mine. Like any good guide, my duty is to make your foray into Android tablet development an enjoyable and fruitful experience. The book you are now holding is the result of the collaborative efforts of many people, and I wish to take this opportunity to acknowledge them here.

First, my personal gratitude to Bob Elliott, executive editor at Wrox. Bob is always ready to lend a listening ear and to offer help when it's needed. It is a great pleasure to work with Bob, as he is one of the most responsive persons I have ever worked with! Thank you, Bob, for the help and guidance!

Of course, I cannot forget Ami Sullivan, my editor (and friend!), who is always a pleasure to work with. After working together on four books, we now know each other so well that we know the content of incoming e-mail messages even before we open them! Thank you, Ami!

Nor can I forget the heroes behind the scenes: copyeditor Luann Rouff and technical editor Chaim Krause. They have been eagle-eye editing the book, making sure that every sentence makes sense — both grammatically and technically. Thanks, Luann and Chaim!

Last, but not least, I want to thank my parents and my wife, Sze Wa, for all the support they have given me. They have selflessly adjusted their schedules to accommodate my busy schedule when I was working on this book. My wife, as always, has stayed up with me on numerous nights as I was furiously working to meet the deadlines, and for this I would like to say to her and my parents, "I love you all!" Finally, to our lovely dog, Ookii, thanks for staying by our side.

CONTENTS

INTRODUCTION

I FIRST STARTED PLAYING WITH THE ANDROID SDK before it was officially released as version 1.0. Back then, the tools were unpolished, the APIs in the SDK were unstable, and the documentation was sparse. Fast-forward three and a half years, Android is now a formidable mobile operating system, with a following no less impressive than the iPhone. Having gone through all the growing pains of Android, I think now is the best time to start learning about Android programming — the APIs have stabilized, and the tools have improved. One challenge remains, however: Getting started is still an elusive goal for many. What's more, Google has recently released their latest version of the Android SDK — 4.0, a unified mobile OS for both smartphones and tablets. The Android 4.0 SDK includes several new features for tablet developers, and understanding all these new features requires some effort on the part of beginners.

It was with this challenge in mind that I was motivated to write this book, one that could benefit beginning Android programmers and enable them to write progressively more sophisticated applications.

As a book written to help jump-start beginning Android developers, it covers the necessary topics in a linear manner so that you can build on your knowledge without being overwhelmed by the details. I adopt the philosophy that the best way to learn is by doing — hence, the numerous Try It Out sections in each chapter, which first show you how to build something and then explain how everything works. I have also taken this opportunity to further improve the previous edition of this book, addressing feedback from readers and adding additional topics that are important to beginning Android developers.

Although Android programming is a huge topic, my aim for this book is threefold: to get you started with the fundamentals, to help you understand the underlying architecture of the SDK, and to appreciate why things are done in certain ways. It is beyond the scope of any book to cover everything under the sun related to Android programming, but I am confident that after reading this book (and doing the exercises), you will be well equipped to tackle your next Android programming challenge.

WHO THIS BOOK IS FOR

This book is targeted for the beginning Android developer who wants to start developing applications using Google's Android SDK. To truly benefit from this book, you should have some background in programming and at least be familiar with object-oriented programming concepts. If you are totally new to Java — the language used for Android development — you might want to take a programming course in Java programming first, or grab one of many good books on Java programming. In my experience, if you already know C# or VB.NET, learning Java is not too much of an effort; you should be comfortable just following along with the Try It Outs.

For those totally new to programming, I know the lure of developing mobile apps and making some money is tempting. However, before attempting to try out the examples in this book, I think a better starting point would be to learn the basics of programming first.

NOTE *All the examples discussed in this book were written and tested using version 4.0 of the Android SDK. While every effort is made to ensure that all the tools used in this book are the latest, it is always possible that by the time you read this book, a newer version of the tools may be available. If so, some of the instructions and/or screenshots may differ slightly. However, any variations should be manageable.*

WHAT THIS BOOK COVERS

This book covers the fundamentals of Android programming using the Android SDK. It is divided into 12 chapters and three appendixes.

Chapter 1: Getting Started with Android Programming covers the basics of the Android OS and its current state. You will learn about the features of Android devices, as well as some of the popular devices on the market. You will also learn how to download and install all the required tools to develop Android applications and then test them on the Android emulator.

Chapter 2: Activities, Fragments, and Intents gets you acquainted with these three fundamental concepts in Android programming. Activities and fragments are the building blocks of an Android application. You will learn how to link activities together to form a complete Android application using intents, one of the unique characteristics of the Android OS.

Chapter 3: Getting to Know the Android User Interface covers the various components that make up the UI of an Android application. You will learn about the various layouts you can use to build the UI of your application, and the numerous events that are associated with the UI when users interact with the application.

Chapter 4: Designing Your User Interface with Views walks you through the various basic views you can use to build your Android UI. You will learn three main groups of views: basic views, picker views, and list views. You will also learn about the specialized fragments available in Android 3.0 and 4.0.

Chapter 5: Displaying Pictures and Menus with Views continues the exploration of views. Here, you will learn how to display images using the various image views, as well as display options and context menus in your application. This chapter ends with some additional cool views that you can use to spice up your application.

Chapter 6: Data Persistence shows you how to save, or store, data in your Android application. In addition to learning the various techniques to store user data, you will also learn file manipulation and how to save files onto internal and external storage (SD card). In addition, you will learn how to create and use a SQLite database in your Android application.

Chapter 7: Content Providers discusses how data can be shared among different applications on an Android device. You will learn how to use a content provider and then build one yourself.

Chapter 8: Messaging explores two of the most interesting topics in mobile programming — sending SMS messages and e-mail. You will learn how to programmatically send and receive SMS and e-mail messages, and how to intercept incoming SMS messages so that the built-in Messaging application will not be able to receive any messages.

Chapter 9: Location-Based Services demonstrates how to build a location-based service application using Google Maps. You will also learn how to obtain geographical location data and then display the location on the map.

Chapter 10: Networking explores how to connect to web servers to download data. You will see how XML and JSON web services can be consumed in an Android application. This chapter also explains sockets programming, and you will learn how to build a chat client in Android.

Chapter 11: Developing Android Services demonstrates how you can write applications using services. Services are background applications that run without a UI. You will learn how to run your services asynchronously on a separate thread, and how your activities can communicate with them.

Chapter 12: Publishing Android Applications discusses the various ways you can publish your Android applications when you are ready. You will also learn about the necessary steps to publishing and selling your applications on the Android Market.

Appendix A: Using Eclipse for Android Development provides a brief overview of the many features in Eclipse.

Appendix B: Using the Android Emulator provides some tips and tricks on using the Android emulator for testing your applications.

Appendix C: Answers to Exercises contains the solutions to the end-of-chapter exercises found in every chapter.

HOW THIS BOOK IS STRUCTURED

This book breaks down the task of learning Android programming into several smaller chunks, enabling you to digest each topic before delving into a more advanced one.

If you are a total beginner to Android programming, start with Chapter 1 first. Once you have familiarized yourself with the basics, head over to the appendixes to read more about Eclipse and the Android emulator. When you are ready, continue with Chapter 2 and gradually move into more advanced topics.

A feature of this book is that all the code samples in each chapter are independent of those discussed in previous chapters. This gives you the flexibility to dive into the topics that interest you and start working on the Try It Out projects.

WHAT YOU NEED TO USE THIS BOOK

All the examples in this book run on the Android emulator (which is included as part of the Android SDK). However, to get the most out of this book, having a real Android device would be useful (though not absolutely necessary).

CONVENTIONS

To help you get the most from the text and keep track of what's happening, a number of conventions are used throughout the book.

TRY IT OUT These Are Exercises or Examples for You to Follow

The Try It Out sections appear once or more per chapter. These are exercises to work through as you follow the related discussion in the text.

1. They consist of a set of numbered steps.
2. Follow the steps with your copy of the project files.

How It Works

After each Try It Out, the code you've typed is explained in detail.

As for other conventions in the text:

- New terms and important words are *highlighted* in italics when first introduced.
- Keyboard combinations are treated like this: Ctrl+R.
- Filenames, URLs, and code within the text are treated like so: `persistence.properties`.
- Code is presented in two different ways:

```
We use a monofont type with no highlighting for most code examples.

We use bolding to emphasize code that is of particular importance in the
present context.
```

NOTE *Notes, tips, hints, tricks, and asides to the current discussion look like this.*

SOURCE CODE

As you work through the examples in this book, you may choose either to type in all the code manually or to use the source code files that accompany the book. All the source code used in this book is available for download at `www.wrox.com`. When at the site, simply locate the book's title (use the Search box or one of the title lists) and click the Download Code link on the book's detail page to obtain all the source code for the book.

You'll find the filename of the project you need in a CodeNote such as this at the beginning of the Try it Out features:

code snippet filename

After you download the code, just decompress it with your favorite compression tool. Alternatively, go to the main Wrox code download page at `www.wrox.com/dynamic/books/download.aspx` to see the code available for this book and for all other Wrox books.

NOTE *Because many books have similar titles, you may find it easiest to search by ISBN; this book's ISBN is 978-1-118-19954-1.*

ERRATA

We make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of our books, such as a spelling mistake or faulty piece of code, we would be very grateful for your feedback. By sending in errata, you may save another reader hours of frustration and at the same time help us provide even higher-quality information.

To find the errata page for this book, go to `www.wrox.com` and locate the title using the Search box or one of the title lists. Then, on the book details page, click the Book Errata link. On this page, you can view all errata that has been submitted for this book and posted by Wrox editors.

NOTE *A complete book list, including links to each book's errata, is also available at* `www.wrox.com/misc-pages/booklist.shtml`.

If you don't spot "your" error on the Book Errata page, go to `www.wrox.com/contact/techsupport.shtml` and complete the form there to send us the error you have found. We'll check the information and, if appropriate, post a message to the book's errata page and fix the problem in subsequent editions of the book.

P2P.WROX.COM

For author and peer discussion, join the P2P forums at `p2p.wrox.com`. The forums are a web-based system for you to post messages relating to Wrox books and related technologies and to interact with other readers and technology users. The forums offer a subscription feature to e-mail you topics of interest of your choosing when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers are present on these forums.

At `p2p.wrox.com`, you will find a number of different forums that will help you not only as you read this book but also as you develop your own applications. To join the forums, just follow these steps:

1. Go to `p2p.wrox.com` and click the Register link.
2. Read the terms of use and click Agree.
3. Complete the required information to join as well as any optional information you want to provide and click Submit.
4. You will receive an e-mail with information describing how to verify your account and complete the joining process.

NOTE *You can read messages in the forums without joining P2P, but in order to post your own messages, you must join.*

After you join, you can post new messages and respond to messages that other users post. You can read messages at any time on the web. If you want to have new messages from a particular forum e-mailed to you, click the Subscribe to This Forum icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works, as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.

1

Getting Started with Android Programming

WHAT YOU WILL LEARN IN THIS CHAPTER

- What is Android?
- Android versions and its feature set
- The Android architecture
- The various Android devices on the market
- The Android Market application store
- How to obtain the tools and SDK for developing Android applications
- How to develop your first Android application

Welcome to the world of Android! When I was writing my first book on Android (which was just less than a year ago), I stated that Android was ranked second in the U.S. smartphone market, second to Research In Motion's (RIM) BlackBerry, and overtaking Apple's iPhone. Shortly after the book went to press, comScore (a global leader in measuring the digital world and the preferred source of digital marketing intelligence) reported that Android has overtaken BlackBerry as the most popular smartphone platform in the U.S.

A few months later, Google released Android 3.0, code named *Honeycomb*. With Android 3.0, Google's focus in the new Software Development Kit was the introduction of several new features

designed for widescreen devices, specifically tablets. If you are writing apps for Android smartphones, Android 3.0 is not really useful, as the new features are not supported on smartphones. At the same time that Android 3.0 was released, Google began working on the next version of Android, which can be
used on both smartphones and tablets. In October 2011, Google released Android 4.0, code named *Ice Cream Sandwich*, and that is the focus of this book.

In this chapter you will learn what Android is, and what makes it so compelling to both developers and device manufacturers alike. You will also get started with developing your first Android application, and learn how to obtain all the necessary tools and set them up so that you can test your application on an Android 4.0 emulator. By the end of this chapter, you will be equipped with the basic knowledge you need to explore more sophisticated techniques and tricks for developing your next killer Android application.

WHAT IS ANDROID?

Android is a mobile operating system that is based on a modified version of Linux. It was originally developed by a startup of the same name, Android, Inc. In 2005, as part of its strategy to enter the mobile space, Google purchased Android and took over its development work (as well as its development team).

Google wanted Android to be open and free; hence, most of the Android code was released under the open source Apache License, which means that anyone who wants to use Android can do so by downloading the full Android source code. Moreover, vendors (typically hardware manufacturers) can add their own proprietary extensions to Android and customize Android to differentiate their products from others. This simple development model makes Android very attractive and has thus piqued the interest of many vendors. This has been especially true for companies affected by the phenomenon of Apple's iPhone, a hugely successful product that revolutionized the smartphone industry. Such companies include Motorola and Sony Ericsson, which for many years have been developing their own mobile operating systems. When the iPhone was launched, many of these manufacturers had to scramble to find new ways of revitalizing their products. These manufacturers see Android as a solution — they will continue to design their own hardware and use Android as the operating system that powers it.

The main advantage of adopting Android is that it offers a unified approach to application development. Developers need only develop for Android, and their applications should be able to run on numerous different devices, as long as the devices are powered using Android. In the world of smartphones, applications are the most important part of the success chain. Device manufacturers therefore see Android as their best hope to challenge the onslaught of the iPhone, which already commands a large base of applications.

Android Versions

Android has gone through quite a number of updates since its first release. Table 1-1 shows the various versions of Android and their codenames.

TABLE 1-1: A Brief History of Android Versions

ANDROID VERSION	RELEASE DATE	CODENAME
1.1	9 February 2009	
1.5	30 April 2009	Cupcake
1.6	15 September 2009	Donut
2.0/2.1	26 October 2009	Eclair
2.2	20 May 2010	Froyo
2.3	6 December 2010	Gingerbread
3.0/3.1/3.2	22 February 2011	Honeycomb
4.0	19 October 2011	Ice Cream Sandwich

In February 2011, Google released Android 3.0, a tablet-only release supporting widescreen devices. The key changes in Android 3.0 are as follows.

- New user interface optimized for tablets
- 3D desktop with new widgets
- Refined multi-tasking
- New web browser features, such as tabbed browsing, form auto-fill, bookmark synchronization, and private browsing
- Support for multi-core processors

Applications written for versions of Android prior to 3.0 are compatible with Android 3.0 devices, and they run without modifications. Android 3.0 tablet applications that make use of the newer features available in 3.0, however, will not be able to run on older devices. To ensure that an Android tablet application can run on all versions of devices, you must programmatically ensure that you only make use of features that are supported in specific versions of Android.

In October 2011, Google released Android 4.0, a version that brought all the features introduced in Android 3.0 to smartphones, along with some new features such as facial recognition unlock, data usage monitoring and control, Near Field Communication (NFC), and more.

Features of Android

Because Android is open source and freely available to manufacturers for customization, there are no fixed hardware or software configurations. However, Android itself supports the following features:

- **Storage** — Uses SQLite, a lightweight relational database, for data storage. Chapter 6 discusses data storage in more detail.
- **Connectivity** — Supports GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth (includes A2DP and AVRCP), Wi-Fi, LTE, and WiMAX. Chapter 8 discusses networking in more detail.

- **Messaging** — Supports both SMS and MMS. Chapter 8 discusses messaging in more detail.
- **Web browser** — Based on the open source WebKit, together with Chrome's V8 JavaScript engine
- **Media support** — Includes support for the following media: H.263, H.264 (in 3GP or MP4 container), MPEG-4 SP, AMR, AMR-WB (in 3GP container), AAC, HE-AAC (in MP4 or 3GP container), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP
- **Hardware support** — Accelerometer Sensor, Camera, Digital Compass, Proximity Sensor, and GPS
- **Multi-touch** — Supports multi-touch screens
- **Multi-tasking** — Supports multi-tasking applications
- **Flash support** — Android 2.3 supports Flash 10.1.
- **Tethering** — Supports sharing of Internet connections as a wired/wireless hotspot

Architecture of Android

In order to understand how Android works, take a look at Figure 1-1, which shows the various layers that make up the Android operating system (OS).

The Android OS is roughly divided into five sections in four main layers:

- **Linux kernel** — This is the kernel on which Android is based. This layer contains all the low-level device drivers for the various hardware components of an Android device.
- **Libraries** — These contain all the code that provides the main features of an Android OS. For example, the SQLite library provides database support so that an application can use it for data storage. The WebKit library provides functionalities for web browsing.
- **Android runtime** — At the same layer as the libraries, the Android runtime provides a set of core libraries that enable developers to write Android apps using the Java programming language. The Android runtime also includes the Dalvik virtual machine, which enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine (Android applications are compiled into Dalvik executables). Dalvik is a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU.
- **Application framework** — Exposes the various capabilities of the Android OS to application developers so that they can make use of them in their applications.
- **Applications** — At this top layer, you will find applications that ship with the Android device (such as Phone, Contacts, Browser, etc.), as well as applications that you download and install from the Android Market. Any applications that you write are located at this layer.

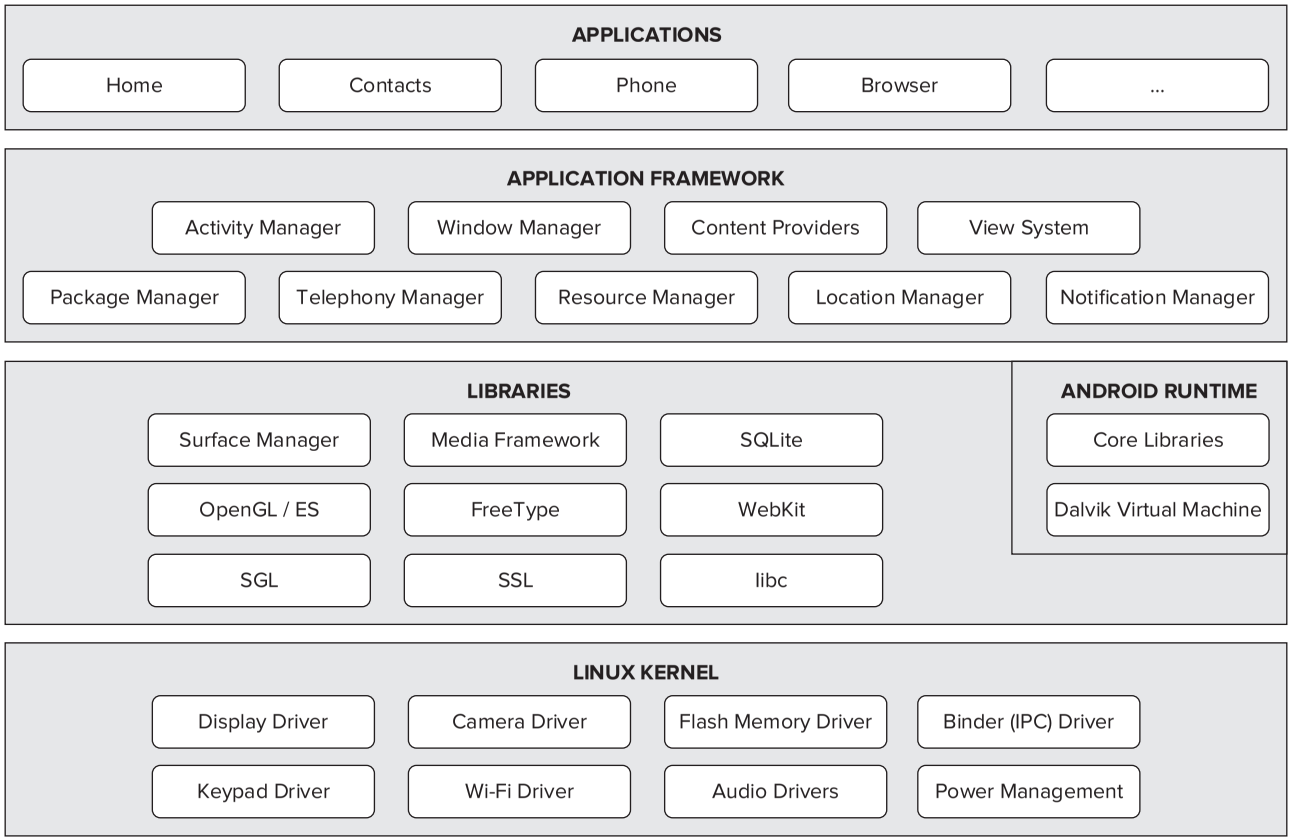

APPLICATIONS
Home
Contacts
Phone
Browser
...
APPLICATION FRAMEWORK
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
LIBRARIES
Surface Manager
Media Framework
SQLite
OpenGL / ES
FreeType
WebKit
SGL
SSL
libc
ANDROID RUNTIME
Core Libraries
Dalvik Virtual Machine
LINUX KERNEL
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
Wi-Fi Driver
Audio Drivers
Power Management

FIGURE 1-1

Android Devices in the Market

Android devices come in all shapes and sizes. As of late November 2011, the Android OS powers the following types of devices:

- Smartphones
- Tablets
- E-reader devices
- Netbooks
- MP4 players
- Internet TVs

Chances are good that you own at least one of the preceding devices. Figure 1-2 shows (left to right) the Samsung Galaxy S II, the Motorola Atrix 4G, and the HTC EVO 4G smartphones.

FIGURE 1-2

Another popular category of devices that manufacturers are rushing out is the *tablet*. Tablets typically come in two sizes: seven inches and ten inches, measured diagonally. Figure 1-3 shows the Samsung Galaxy Tab 10.1 (left) and the Asus Eee Pad Transformer TF101 (right), both 10.1-inch tablets. Both the Samsung Galaxy 10.1 and the Asus Eee Pad Transfer TF101 run on Android 3.

FIGURE 1-3

Besides smartphones and tablets, Android is also beginning to appear in dedicated devices, such as e-book readers. Figure 1-4 shows the Barnes and Noble's NOOK Color (left) and Amazon's Kindle Fire (right), both of which are color e-Book readers running the Android OS.

FIGURE 1-4

In addition to these popular mobile devices, Android is also slowly finding its way into your living room. People of Lava, a Swedish company, has developed an Android-based TV, called the Scandinavia Android TV (see Figure 1-5).

Google has also ventured into a proprietary smart TV platform based on Android and codeveloped with companies such as Intel, Sony, and Logitech. Figure 1-6 shows Sony's Google TV.

FIGURE 1-5

FIGURE 1-6

At the time of writing, the Samsung Galaxy Nexus (see Figure 1-7) is the only device running on Android 4.0. However, Google has promised that existing devices (such as the Nexus S) will be able to upgrade to Android 4.0. By the time you are reading this, there should be a plethora of devices running Android 4.0.

FIGURE 1-7

The Android Market

As mentioned earlier, one of the main factors determining the success of a smartphone platform is the applications that support it. It is clear from the success of the iPhone that applications play a very vital role in determining whether a new platform swims or sinks. In addition, making these applications accessible to the general user is extremely important.

As such, in August 2008, Google announced Android Market, an online application store for Android devices, and made it available to users in October 2008. Using the Market application that is preinstalled on their Android device, users can simply download third-party applications directly onto their devices. Both paid and free applications are supported on the Android Market, though paid applications are available only to users in certain countries due to legal issues.

Similarly, in some countries, users can buy paid applications from the Android Market, but developers cannot sell in that country. As an example, at the time of writing, users in India can buy apps from the Android Market, but developers in India cannot sell apps on the Android Market. The reverse may also be true; for example, users in South Korea cannot buy apps, but developers in South Korea can sell apps on the Android Market.

NOTE *Chapter 12 discusses more about the Android Market and how you can sell your own applications in it.*

The Android Developer Community

With Android in its fourth version, there is a large developer community all over the world. It is now much easier to get solutions to problems, and find like-minded developers to share app ideas and exchange experiences.

Here are some developer communities/sites that you can turn to for help if you run into problems while working with Android:

- **Stack Overflow** (`www.stackoverflow.com`) — Stack Overflow is a collaboratively edited question and answer site for developers. If you have a question about Android, chances are someone at Stack Overflow is probably already discussing the same question and someone else had already provided the answer. Best of all, other developers can vote for the best answer so that you can know which are the answers that are trustworthy.
- **Google Android Training** (`http://developer.android.com/training/index.html`) — Google has launched the Android Training site that contains a number of useful classes grouped by topics. At the time of writing, the classes mostly contain useful code snippets that are very useful to Android developers once they have started with the basics. Once you have learned the basics in this book, I strongly suggest you take a look at the classes.
- **Android Discuss** (`http://groups.google.com/group/android-discuss`) — Android Discuss is a discussion group hosted by Google using the Google Groups service. Here, you will be able to discuss the various aspects of Android programming. This group is monitored closely by the Android team at Google, and so this is good place to clarify your doubts and learn new tips and tricks.

OBTAINING THE REQUIRED TOOLS

Now that you know what Android is and what its feature set contains, you are probably anxious to get your hands dirty and start writing some applications! Before you write your first app, however, you need to download the required tools and SDKs.

For Android development, you can use a Mac, a Windows PC, or a Linux machine. All the tools needed are free and can be downloaded from the Web. Most of the examples provided in this book should work fine with the Android emulator, with the exception of a few examples that require access to the hardware. For this book, I am using a Windows 7 computer to demonstrate all the code samples. If you are using a Mac or Linux computer, the screenshots should look similar; some minor differences may be present, but you should be able to follow along without problems.

Let the fun begin!

JAVA JDK

The Android SDK makes use of the Java SE Development Kit (JDK). If your computer does not have the JDK installed, you should start by downloading it from `www.oracle.com/technetwork/java/javase/downloads/index.html` and installing it prior to moving to the next section.

Android SDK

The first and most important piece of software you need to download is, of course, the Android SDK. The Android SDK contains a debugger, libraries, an emulator, documentation, sample code, and tutorials.

You can download the Android SDK from `http://developer.android.com/sdk/index.html` (see Figure 1-8).

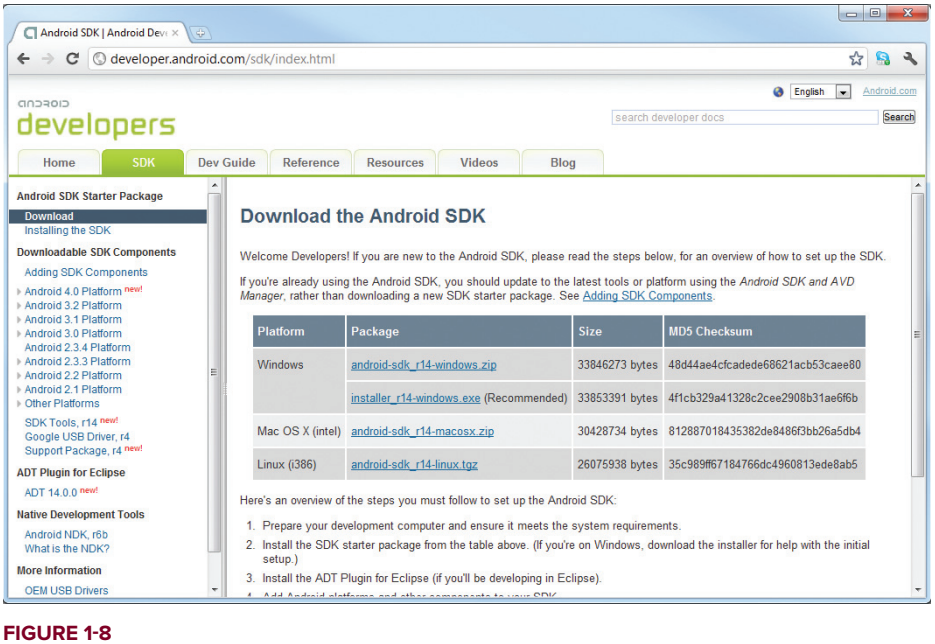


FIGURE 1-8

The Android SDK is packaged in a zip file. You can download it and unzip its content (the `android-sdk-windows` folder) into a folder, say `C:\Android 4.0\`. For Windows user, Google recommends that you download the `installer_r15-windows.exe` file instead and use it to set up the tools for you automatically. The following steps walk you through the installation process using this approach.

Installing the Android SDK Tools

When you have downloaded the `installer_r15-windows.exe` file, double-click it to start the installation of the Android tools. In the welcome screen of the Setup Wizard, click Next to continue.

If your computer does not have Java installed, you will see the error dialog shown in Figure 1-9. However, even if you have Java installed, you may still see this error. If this is the case, click the Report error button and then click Next.

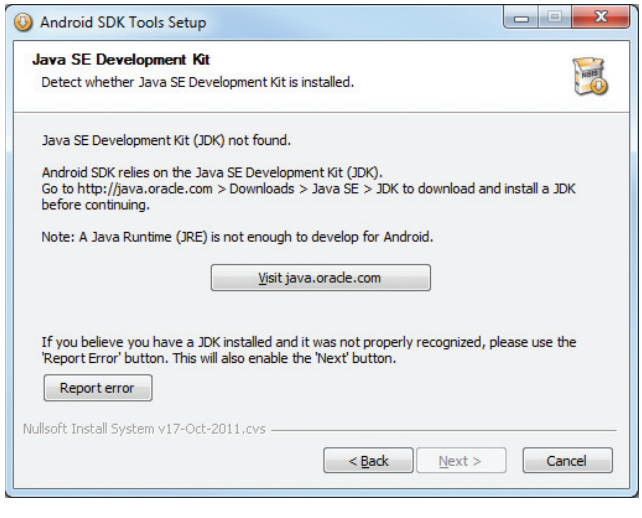


FIGURE 1-9

You will be asked to provide a destination folder to install the Android SDK tools. Enter a destination path (see Figure 1-10) and click Next.

When you are asked to choose a Start Menu folder to create the program's shortcut, take the default "Android SDK Tools" and click Install. When the setup is done, check the "Start SDK Manager (to download system images, etc.)" option and click Finish (see Figure 1-11). This will start the SDK Manager.

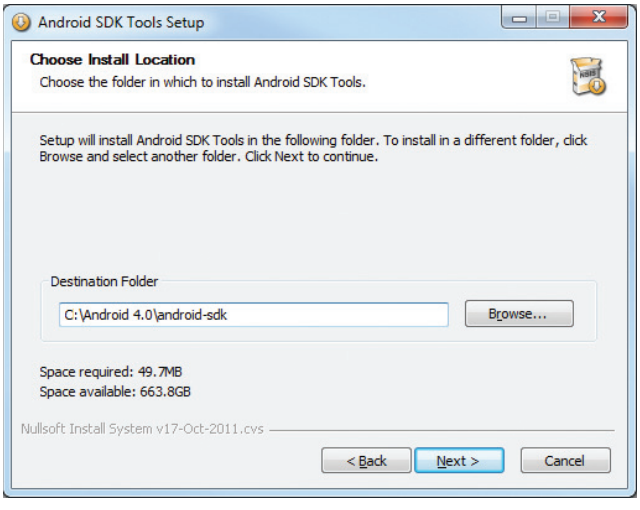


FIGURE 1-10

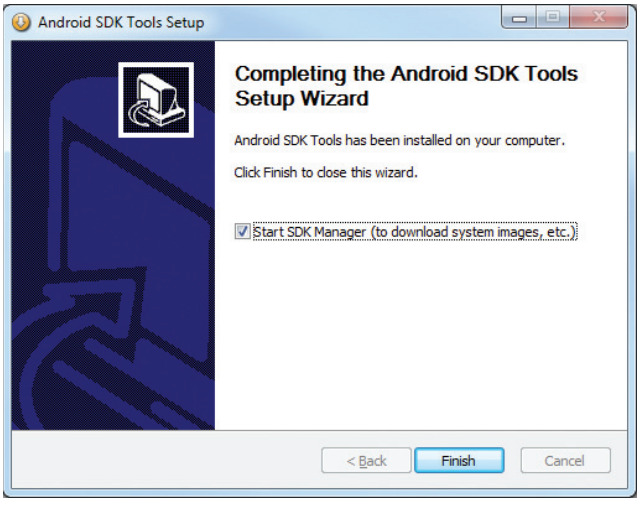


FIGURE 1-11

Configuring the Android SDK Manager

The Android SDK Manager manages the various versions of the Android SDK currently installed on your computer. When it is launched, you will see a list of items and whether or not they are currently installed on your computer (see Figure 1-12).

Check the relevant tools, documentation, and platforms you need for your project. Once you have selected the items you want, click the Install button to download them. Because it takes a while to download from Google's server, it is a good idea to download only what you need immediately, and download the rest when you have more time. For now, you may want to check the items shown in the figure.

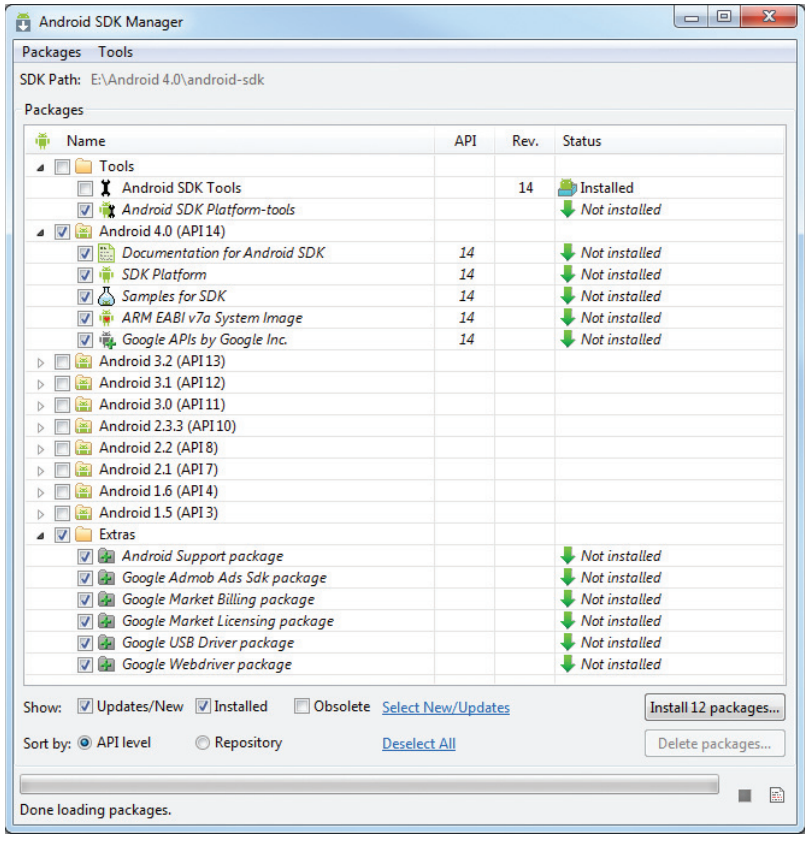


FIGURE 1-12

NOTE *For a start, you should at least select the latest Android 4.0 SDK platform and the Extras. At the time of writing, the latest SDK platform is SDK Platform Android 4.0, API 14.*

Each version of the Android OS is identified by an API level number. For example, Android 2.3.3 is level 10 (API 10), while Android 3.0 is level 11 (API 11), and so on. For each level, two platforms are available. For example, level 14 offers the following:

- SDK Platform
- Google APIs by Google Inc.

The key difference between the two is that the Google APIs platform contains additional APIs provided by Google (such as the Google Maps library). Therefore, if the application you are writing requires Google Maps, you need to create an AVD using the Google APIs platform (more on this is provided in Chapter 9, "Location-Based Services."

You will be asked to choose the packages to install (see Figure 1-13). Check the Accept All option and click Install.

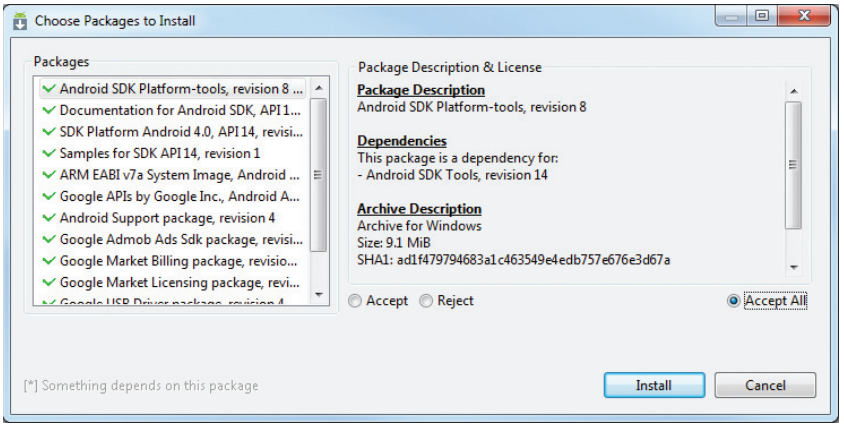


FIGURE 1-13

The SDK Manager will proceed to download the packages that you have selected. The installation takes some time, so be patient. When all the packages are installed, you will be asked to restart the ADB (Android Debug Bridge). Click Yes.

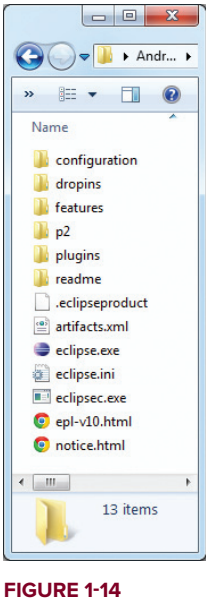


FIGURE 1-14

Eclipse

The next step is to obtain the integrated development environment (IDE) for developing your Android applications. In the case of Android, the recommended IDE is Eclipse, a multi-language software development environment featuring an extensible plug-in system. It can be used to develop various types of applications, using languages such as Java, Ada, C, C++, COBOL, Python, and others.

For Android development, you should download the Eclipse IDE for Java EE Developers (`www.eclipse.org/downloads/`). Six editions are available: Windows (32- and 64-bit), Mac OS X (Cocoa 32- and 64), and Linux (32- and 64-bit). Simply select the relevant one for your operating system. All the examples in this book were tested using the 32-bit version of Eclipse for Windows.

Once the Eclipse IDE is downloaded, unzip its content (the `eclipse` folder) into a folder, say `C:\Android 4.0\`. Figure 1-14 shows the content of the `eclipse` folder.

To launch Eclipse, double-click on the `eclipse.exe` file. You are first asked to specify your workspace. In Eclipse, a workspace is a folder where you store all your projects. Take the default suggested (or you can specify your own folder as the workspace) and click OK.

Android Development Tools (ADT)

When Eclipse is launched, select Help ➪ Install New Software (see Figure 1-15) to install the Android Development Tools (ADT) plug-in for Eclipse.

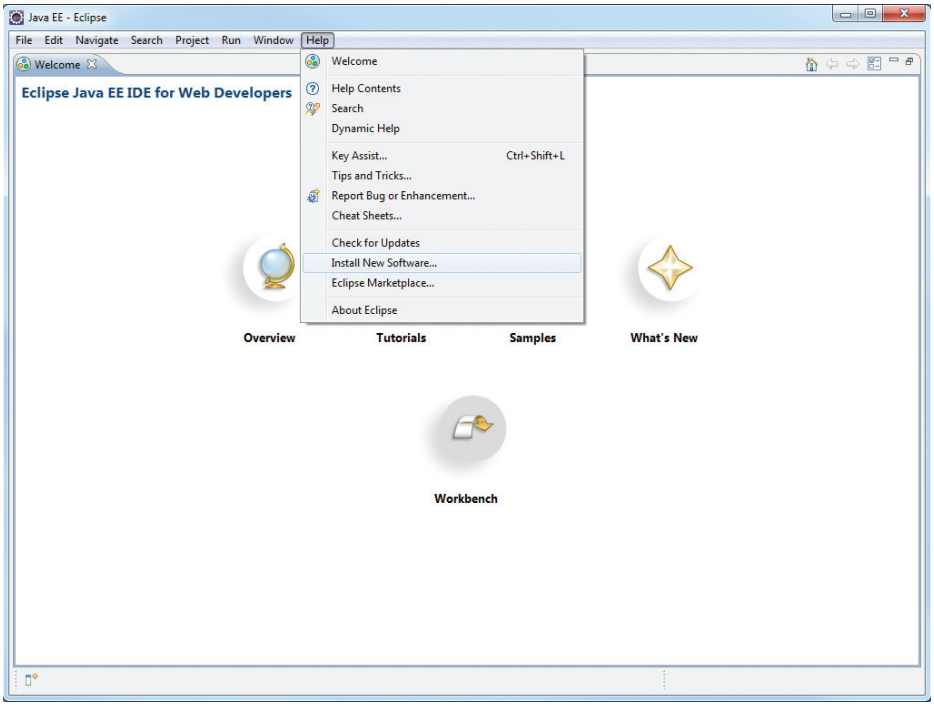


FIGURE 1-15

The ADT is an extension to the Eclipse IDE that supports the creation and debugging of Android applications. Using the ADT, you will be able to do the following in Eclipse:

- Create new Android application projects.
- Access the tools for accessing your Android emulators and devices.
- Compile and debug Android applications.
- Export Android applications into Android Packages (APKs).
- Create digital certificates for code-signing your APK.

In the Install dialog that appears, specify `https://dl-ssl.google.com/android/eclipse/` and press Enter. After a while, you will see the Developer Tools item appear in the middle of the window (see Figure 1-16). Expand it to reveal its content: Android DDMS, Android Development Tools, Android Hierarchy Viewer, and Android Traceview. Check all of them and click Next twice.

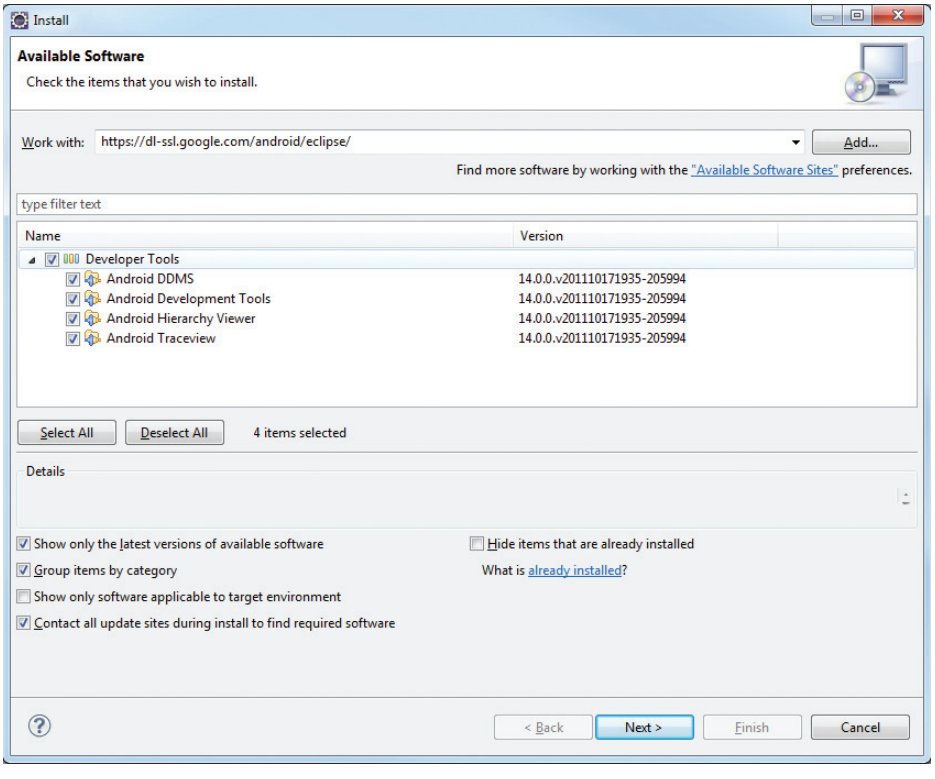


FIGURE 1-16

NOTE *If you have any problems downloading the ADT, check out Google's help at* `http://developer.android.com/sdk/eclipse-adt.html#installing`.

You will be asked to review and accept the licenses. Check the "I accept the terms of the license agreements" option and click Finish. Once the installation is completed, you will be asked to restart Eclipse. Go ahead and restart Eclipse now.

When Eclipse is restarted, you are asked to configure your Android SDK (see Figure 1-17). As the Android SDK has already been downloaded earlier in the previous section, check the "Use existing SDKs" option and specify the directory where you have installed the Android SDK. Click Next.

After this step, you are asked to send your usage statistics to Google. Once you have selected your choice, click Finish.

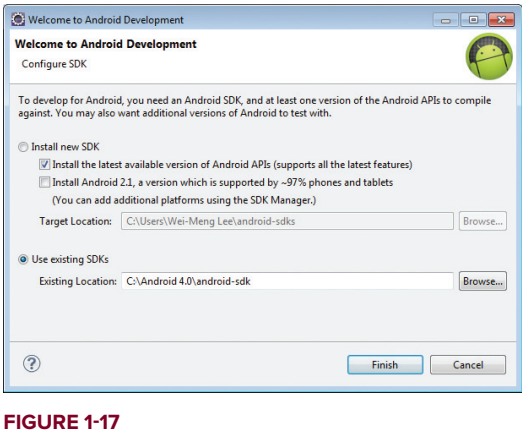


FIGURE 1-17

NOTE *As each new version of the SDK is released, the installation steps tend to differ slightly. If you do not experience the same steps as described here, don't worry — just follow the instructions on screen.*

Creating Android Virtual Devices (AVDs)

The next step is to create an Android Virtual Device (AVD) to be used for testing your Android applications. An AVD is an emulator instance that enables you to model an actual device. Each AVD consists of a hardware profile; a mapping to a system image; as well as emulated storage, such as a secure digital (SD) card.

You can create as many AVDs as you want in order to test your applications with several different configurations. This testing is important to confirm the behavior of your application when it is run on different devices with varying capabilities.

NOTE *Appendix B discusses some of the capabilities of the Android emulator.*

To create an AVD, select Window ➪ AVD Manager (see Figure 1-18).

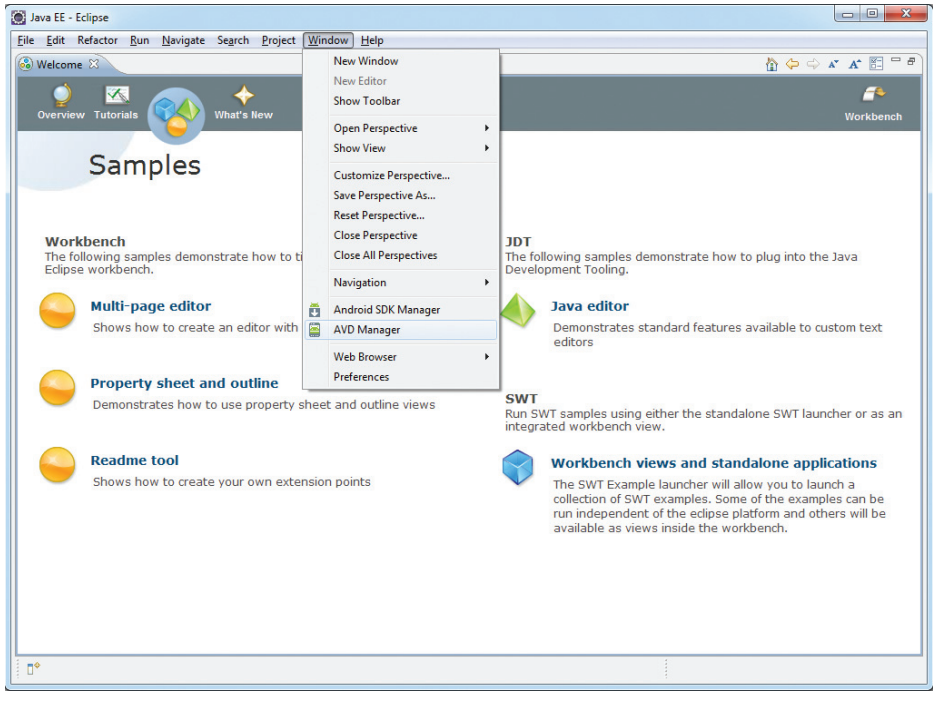


FIGURE 1-18

In the Android Virtual Device Manager dialog (see Figure 1-19), click the New... button to create a new AVD.

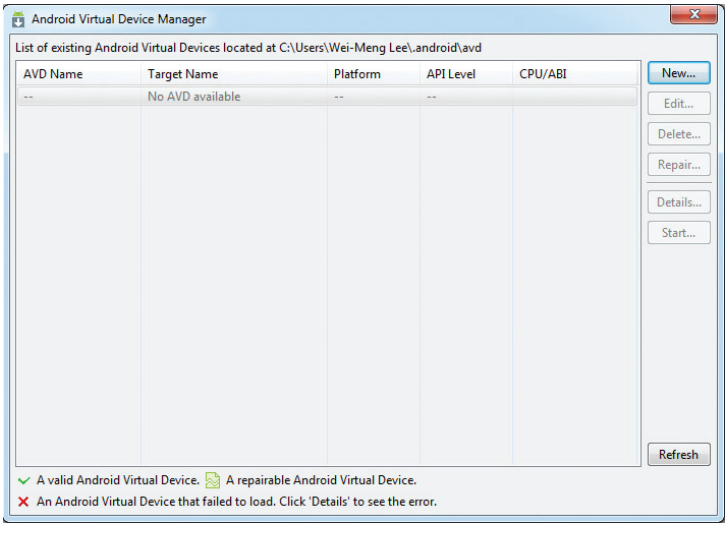


FIGURE 1-19

In the Create new Android Virtual Device (AVD) dialog, enter the items as shown in Figure 1-20. Click the Create AVD button when you are done.

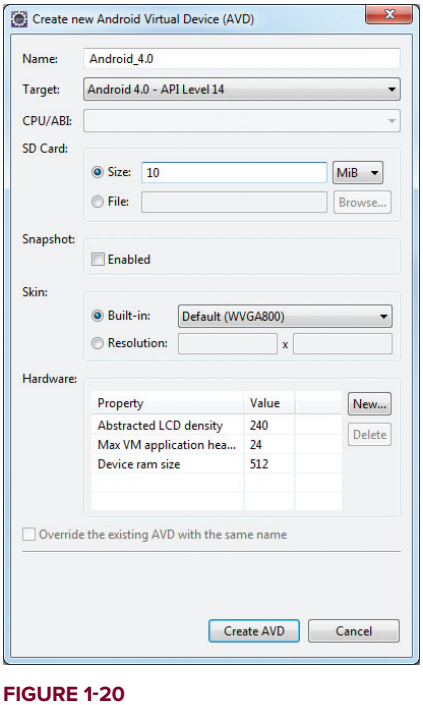


FIGURE 1-20

In this case, you have created an AVD (put simply, an Android emulator) that emulates an Android device running version 4.0 of the OS with a built-in 10-MB SD card. In addition to what you have created, you also have the option to emulate the device with different screen densities and resolutions.

NOTE *Appendix B explains how to emulate the different types of Android devices.*

It is preferable to create a few AVDs with different API levels and hardware configurations so that your application can be tested on different versions of the Android OS.

Once your ADV has been created, it is time to test it. Select the AVD that you want to test and click the Start… button. The Launch Options dialog will appear (see Figure 1-21). If you have a small monitor, it is recommended that you check the "Scale display to real size" option so that you can set the emulator to a smaller size. Click the Launch button to start the emulator.

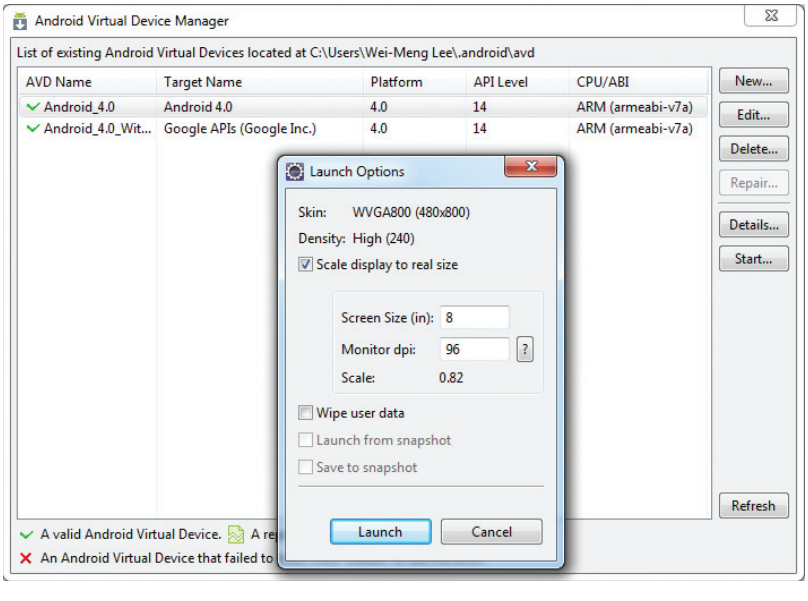


FIGURE 1-21

The Android emulator will start, and after a while it will be ready for use (see Figure 1-22). Go ahead and try out the emulator. It will behave just like a real Android device. After that, in the next section you will learn how to write your first Android application!

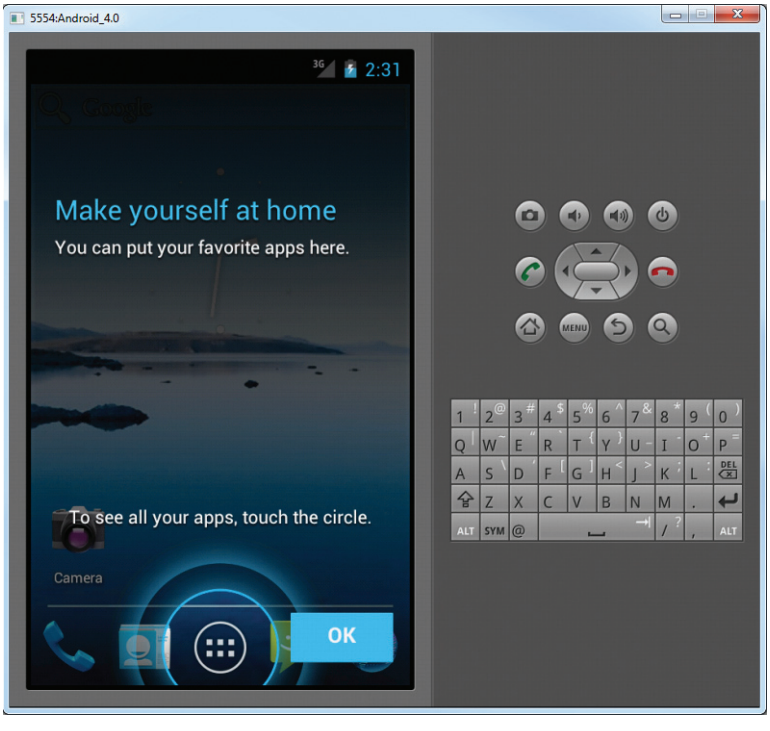


FIGURE 1-22

CREATING YOUR FIRST ANDROID APPLICATION

With all the tools and the SDK downloaded and installed, it is now time to start your engine. As in all programming books, the first example uses the ubiquitous Hello World application. This will give you a detailed look at the various components that make up an Android project.

TRY IT OUT Creating Your First Android Application

codefile HelloWorld.zip available for download at Wrox.com

1. Using Eclipse, create a new project by selecting File ➪ New ➪ Project . . . (see Figure 1-23).

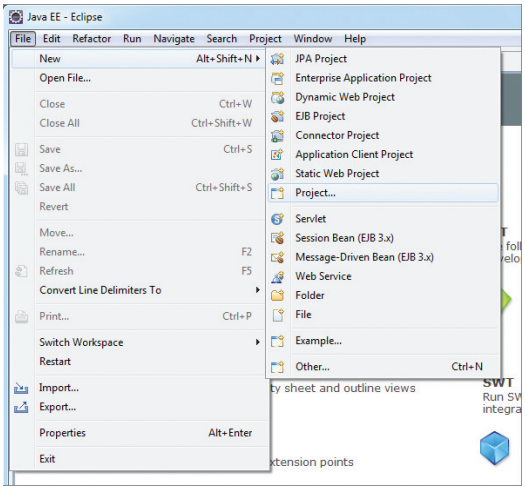


FIGURE 1-23

NOTE *After you have created your first Android application, subsequent Android projects can be created by selecting File ➪ New ➪ Android Project.*

2. Expand the Android folder and select Android Project (see Figure 1-24). Click Next.
3. Name the Android project **HelloWorld**, as shown in Figure 1-25, and then click Next.

FIGURE 1-24

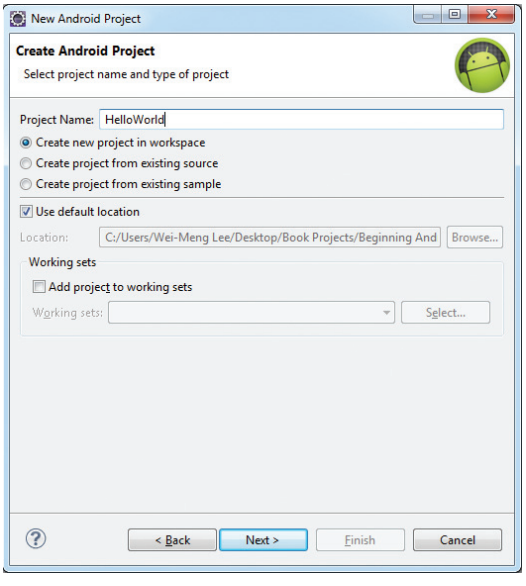


FIGURE 1-25

4. Select the Android 4.0 target and click Next.
5. Fill in the Application Info details as shown in Figure 1-26. Click Finish.

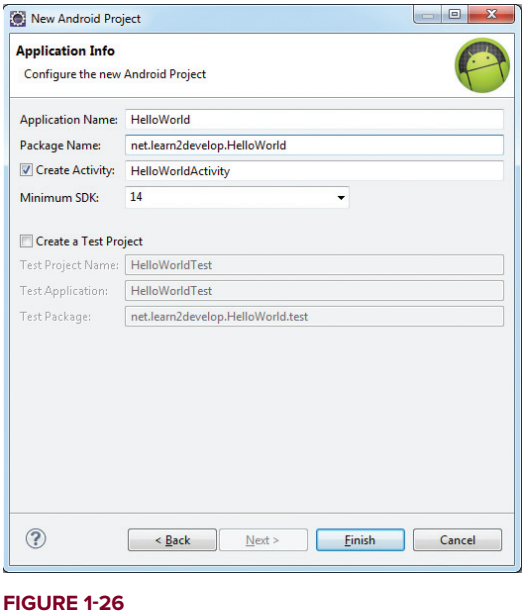


FIGURE 1-26

NOTE *You need to have at least a period (.) in the package name. The recommended convention for the package name is to use your domain name in reverse order, followed by the project name. For example, my company's domain name is* `learn2develop.net`*; hence, my package name would be* `net.learn2develop.HelloWorld`*.*

6. The Eclipse IDE should now look like Figure 1-27.

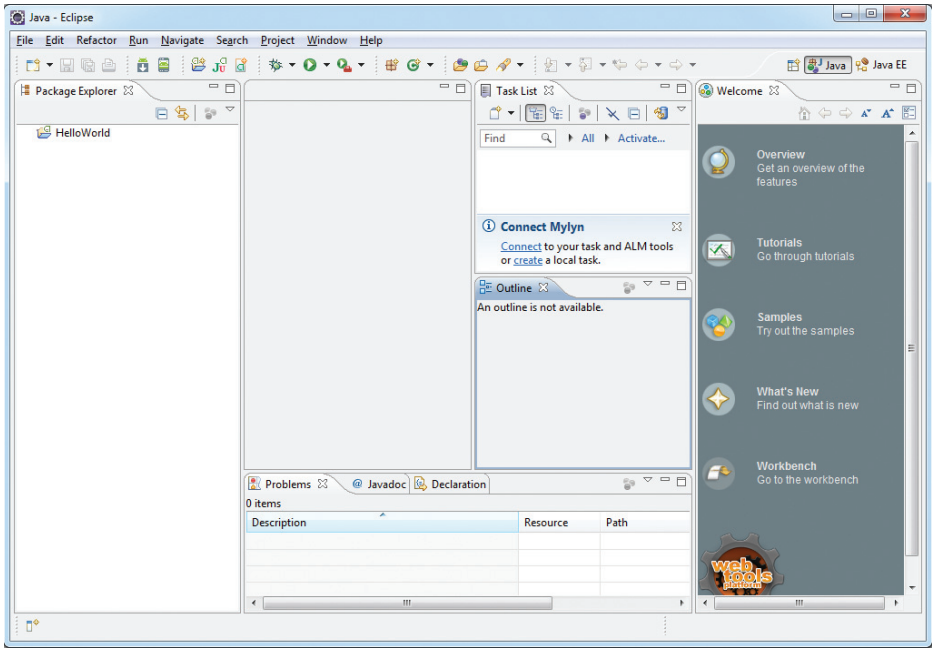


FIGURE 1-27

7. In the Package Explorer (located on the left of the Eclipse IDE), expand the HelloWorld project by clicking on the various arrows displayed to the left of each item in the project (see Figure 1-28). In the `res/layout` folder, double-click the `main.xml` file.

8. The `main.xml` file defines the user interface (UI) of your application. The default view is the Layout view, which lays out the activity graphically. To modify the UI by hand, click the `main.xml` tab located at the bottom (see Figure 1-29).

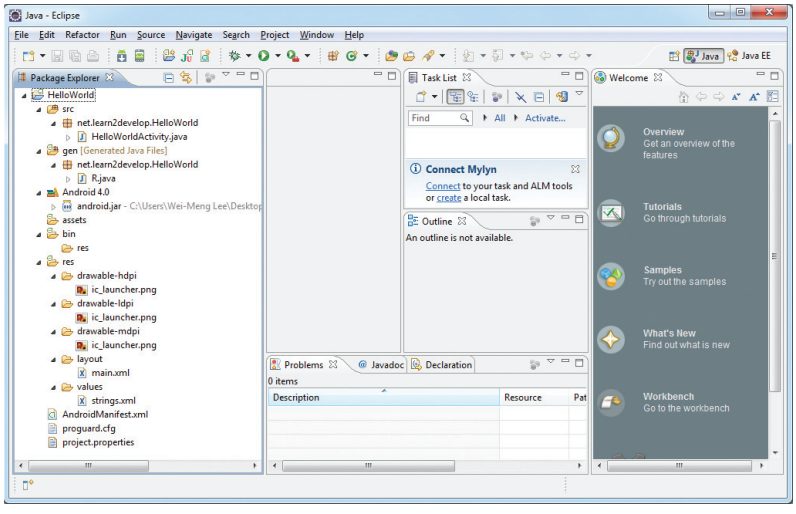


FIGURE 1-28

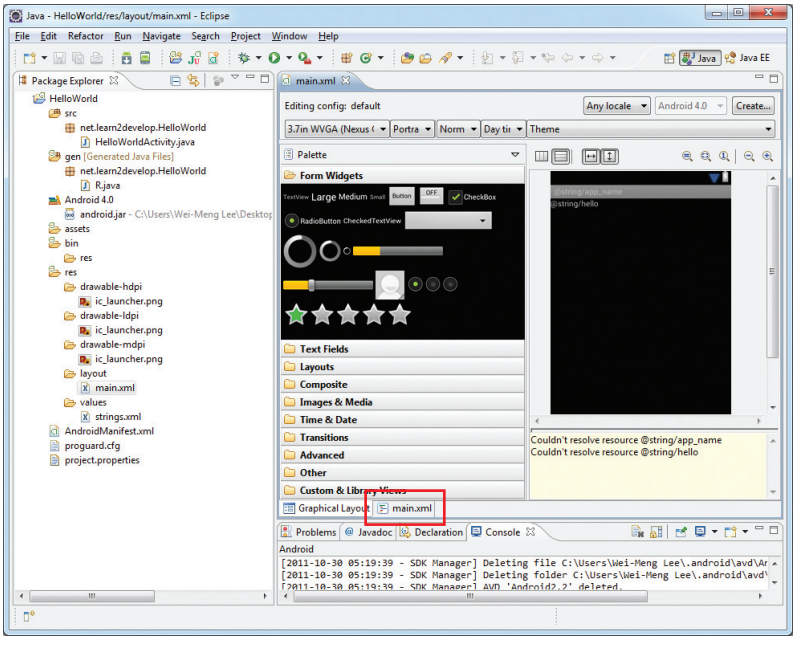


FIGURE 1-29

9. Add the following code in bold to the `main.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent"
    android:orientation="vertical" >

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

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="This is my first Android Application!" />

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="And this is a clickable button!" />

</LinearLayout>
```

10. To save the changes made to your project, press Ctrl+S.

11. You are now ready to test your application on the Android emulator. Right-click the project name in Eclipse and select Run As ➪ Android Application (see Figure 1-30).

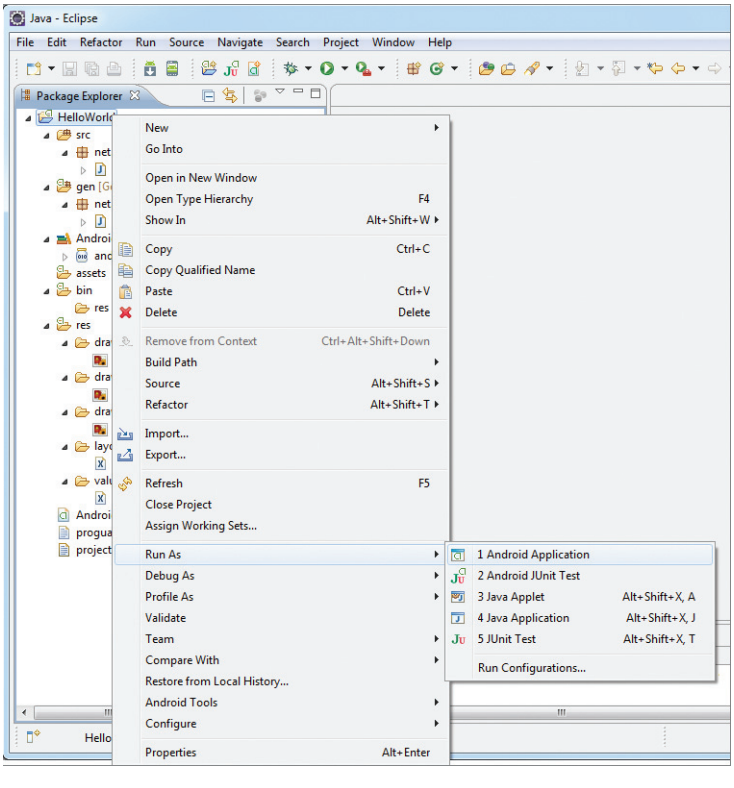


FIGURE 1-30

12. If you have not made any mistakes in the project, you should now be able to see the application installed and running on the Android emulator (see Figure 1-31).

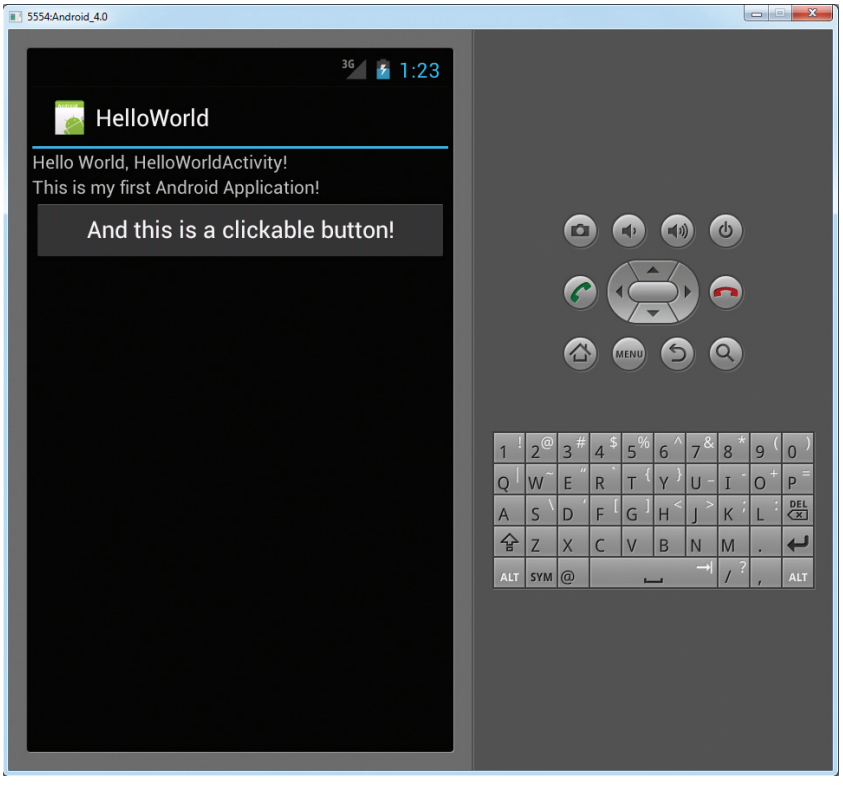


FIGURE 1-31

13. Click the Home button (the house icon in the lower-left corner above the keyboard) so that it now shows the Home screen (see Figure 1-32).

14. Click the application launcher icon to display the list of applications installed on the device. Note that the HelloWorld application is now installed in the application launcher (see Figure 1-33).

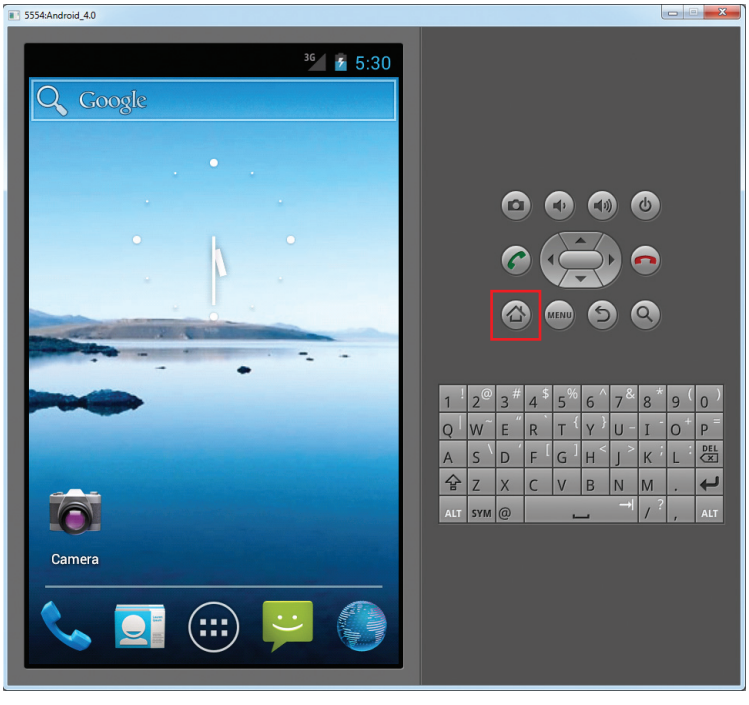


FIGURE 1-32

FIGURE 1-33

WHICH AVD WILL BE USED TO TEST YOUR APPLICATION?

Recall that earlier you created a few AVDs using the AVD Manager. So which one will be launched by Eclipse when you run an Android application? Eclipse checks the target that you specified (when you created a new project), comparing it against the list of AVDs that you have created. The first one that matches will be launched to run your application.

If you have more than one suitable AVD running prior to debugging the application, Eclipse will display the Android Device Chooser dialog, which enables you to select the desired emulator/device to debug the application (see Figure 1-34).

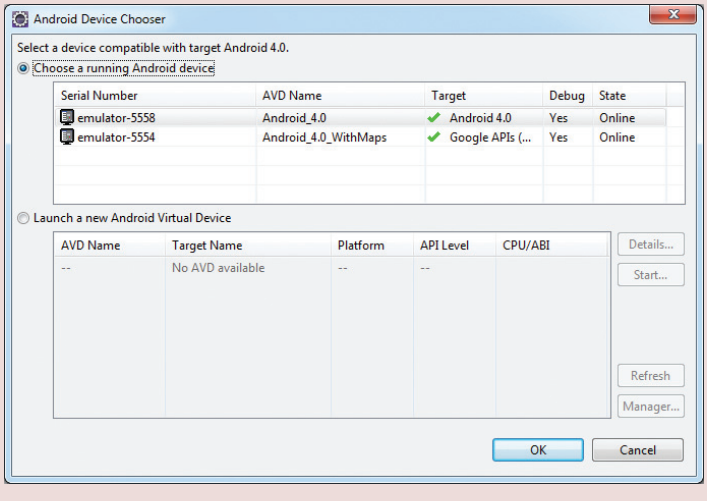


FIGURE 1-34

How It Works

To create an Android project using Eclipse, you need to supply the information shown in Table 1-2.

TABLE 1-2: Project Files Created by Default

PROPERTIES	DESCRIPTION
Project name	The name of the project
Application name	A user-friendly name for your application
Package name	The name of the package. You should use a reverse domain name for this.
Create Activity	The name of the first activity in your application
Min SDK Version	The minimum version of the SDK that your project is targeting

In Android, an *activity* is a window that contains the user interface of your applications. An application can have zero or more activities; in this example, the application contains one activity: `HelloWorldActivity`. This `HelloWorldActivity` is the entry point of the application, which is displayed when the application is started. Chapter 2 discusses activities in more detail.

In this simple example, you modified the `main.xml` file to display the string "This is my first Android Application!" and a button. The `main.xml` file contains the user interface of the activity, which is displayed when `HelloWorldActivity` is loaded.

When you debug the application on the Android emulator, the application is automatically installed on the emulator. And that's it — you have developed your first Android application!

The next section unravels how all the various files in your Android project work together to make your application come alive.

ANATOMY OF AN ANDROID APPLICATION

Now that you have created your first Hello World Android application, it is time to dissect the innards of the Android project and examine all the parts that make everything work.

First, note the various files that make up an Android project in the Package Explorer in Eclipse (see Figure 1-35).

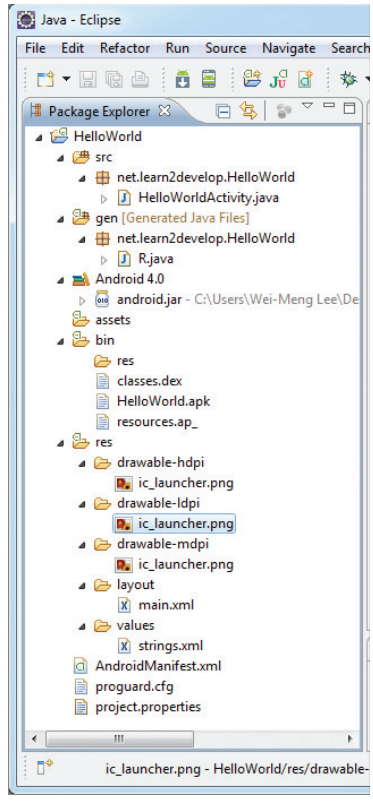


FIGURE 1-35

The various folders and their files are as follows:

- `src` — Contains the `.java` source files for your project. In this example, there is one file, `HelloWorldActivity.java`. The `HelloWorldActivity.java` file is the source file for your activity. You write the code for your application in this file. The Java file is listed under the package name for your project, which in this case is `net.learn2develop.HelloWorld`.
- `gen` — Contains the `R.java` file, a compiler-generated file that references all the resources found in your project. You should not modify this file. All the resources in your project are automatically compiled into this class so that you can refer to them using the class.
- `Android 4.0` library — This item contains one file, `android.jar`, which contains all the class libraries needed for an Android application.
- `assets` — This folder contains all the assets used by your application, such as HTML, text files, databases, etc.
- `bin` — This folder contains the files built by the ADT during the build process. In particular, it generates the `.apk` file (Android Package). An `.apk` file is the application binary of an Android application. It contains everything needed to run an Android application.

- `res` — This folder contains all the resources used in your application. It also contains a few other subfolders: `drawable-<resolution>`, `layout`, and `values`. Chapter 3 talks more about how you can support devices with different screen resolutions and densities.
- `AndroidManifest.xml` — This is the manifest file for your Android application. Here you specify the permissions needed by your application, as well as other features (such as intent-filters, receivers, etc.). Chapter 2 discusses the use of the `AndroidManifest.xml` file in more detail.

The `main.xml` file defines the user interface for your activity. Observe the following in bold:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello" />
```

The `@string` in this case refers to the `strings.xml` file located in the `res/values` folder. Hence, `@string/hello` refers to the `hello` string defined in the `strings.xml` file, which is "Hello World, HelloWorldActivity!":

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="hello">Hello World, HelloWorldActivity!</string>
    <string name="app_name">HelloWorld</string>

</resources>
```

It is recommended that you store all the string constants in your application in this `strings.xml` file and reference these strings using the `@string` identifier. That way, if you ever need to localize your application to another language, all you need to do is make a copy of the entire `values` folder and modify the values of `strings.xml` to contain the string in the language that you want to display. Figure 1-36 shows that I have another folder named `values-fr` with the `strings.xml` file containing the same `hello` string in French.

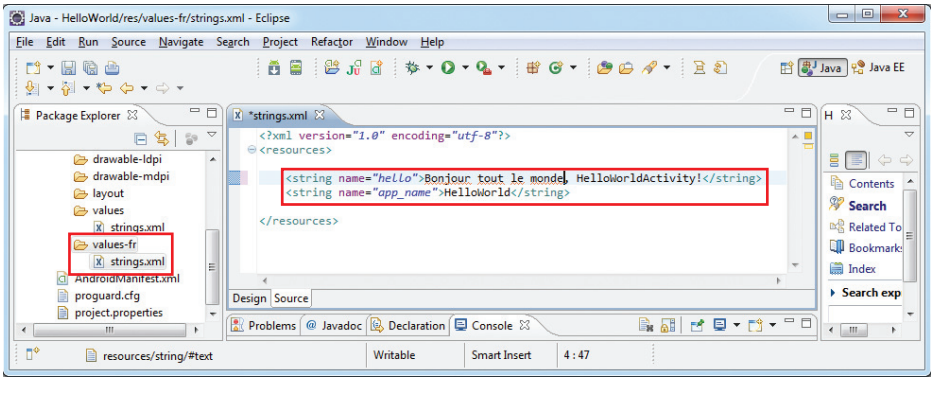


FIGURE 1-36

If the user loads the same application on a phone configured to display French as the default language, your application will automatically display the `hello` string in French.

The next important file in an Android project is the manifest file. Note the content of the `AndroidManifest.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.HelloWorld"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".HelloWorldActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

The `AndroidManifest.xml` file contains detailed information about the application:

- It defines the package name of the application as `net.learn2develop.HelloWorld`.
- The version code of the application is 1 (set via the `android:versionCode` attribute). This value is used to identify the version number of your application. It can be used to programmatically determine whether an application needs to be upgraded.
- The version name of the application is 1.0 (set via the `android:versionName` attribute). This string value is mainly used for display to the user. You should use the format *<major>.<minor>.<point>* for this value.
- The `android:minSdkVersion` attribute of the `<uses-sdk>` element specifies the minimum version of the OS on which the application will run.
- The application uses the image named `ic_launcher.png` located in the `drawable` folders.
- The name of this application is the string named `app_name` defined in the `strings.xml` file.
- There is one activity in the application represented by the `HelloWorldActivity.java` file. The label displayed for this activity is the same as the application name.

- Within the definition for this activity, there is an element named `<intent-filter>`:
  - The action for the intent filter is named `android.intent.action.MAIN` to indicate that this activity serves as the entry point for the application.
  - The category for the intent-filter is named `android.intent.category.LAUNCHER` to indicate that the application can be launched from the device's launcher icon. Chapter 2 discusses intents in more detail.

As you add more files and folders to your project, Eclipse will automatically generate the content of `R.java`, which currently contains the following:

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package net.learn2develop.HelloWorld;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

You are not supposed to modify the content of the `R.java` file; Eclipse automatically generates the content for you when you modify your project.

NOTE *If you delete* `R.java` *manually, Eclipse will regenerate it for you immediately. Note that in order for Eclipse to generate the* `R.java` *file for you, the project must not contain any errors. If you realize that Eclipse has not regenerated* `R.java` *after you have deleted it, check your project again. The code may contain syntax errors, or your XML files (such as* `AndroidManifest.xml`, `main.xml`, *etc.) may not be well-formed.*

Finally, the code that connects the activity to the UI (`main.xml`) is the `setContentView()` method, which is in the `HelloWorldActivity.java` file:

```
package net.learn2develop.HelloWorld;

import android.app.Activity;
import android.os.Bundle;

public class HelloWorldActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Here, `R.layout.main` refers to the `main.xml` file located in the `res/layout` folder. As you add additional XML files to the `res/layout` folder, the filenames will automatically be generated in the `R.java` file. The `onCreate()` method is one of many methods that are fired when an activity is loaded. Chapter 2 discusses the life cycle of an activity in more detail.

SUMMARY

This chapter has provided a brief overview of Android, and highlighted some of its capabilities. If you have followed the sections on downloading the tools and the Android SDK, you should now have a working system — one that is capable of developing more interesting Android applications other than the Hello World application. In the next chapter, you will learn about the concepts of activities and intents, and the very important roles they play in Android.

EXERCISES

1. What is an AVD?
2. What is the difference between the `android:versionCode` and `android:versionName` attributes in the `AndroidManifest.xml` file?
3. What is the use of the `strings.xml` file?

Answers to the exercises can be found in Appendix C.

▶ WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
Android OS	Android is an open source mobile operating system based on the Linux operating system. It is available to anyone who wants to adapt it to run on their own devices.
Languages used for Android application development	You use the Java programming language to develop Android applications. Written applications are compiled into Dalvik executables, which are then run on top of the Dalvik virtual machine.
Android Market	The Android Market hosts all the various Android applications written by third-party developers.
Tools for Android application development	Eclipse IDE, Android SDK, and the ADT
Activities	An activity is represented by a screen in your Android application. Each application can have zero or more activities.
The Android manifest file	The `AndroidManifest.xml` file contains detailed configuration information for your application. As your example application becomes more sophisticated, you will modify this file, and you will see the different information you can add to it as you progress through the chapters.

2

Activities, Fragments, and Intents

WHAT YOU WILL LEARN IN THIS CHAPTER

- The life cycles of an activity
- Using fragments to customize your UI
- Applying styles and themes to activities
- How to display activities as dialog windows
- Understanding the concept of intents
- Using the Intent object to link activities
- How intent filters help you selectively connect to other activities
- Displaying alerts to the user using notifications

In Chapter 1, you learned that an activity is a window that contains the user interface of your application. An application can have zero or more activities. Typically, applications have one or more activities; and the main purpose of an activity is to interact with the user. From the moment an activity appears on the screen to the moment it is hidden, it goes through a number of stages, known as an activity's *life cycle*. Understanding the life cycle of an activity is vital to ensuring that your application works correctly. In addition to activities, Android 4.0 also supports a feature that was introduced in Android 3.0 (for tablets): fragments. Think of fragments as "miniature" activities that can be grouped to form an activity. In this chapter, you will learn about how activities and fragments work together.

Apart from activities, another unique concept in Android is that of an *intent*. An intent is basically the "glue" that enables different activities from different applications to work together seamlessly, ensuring that tasks can be performed as though they all belong to one single application. Later in this chapter, you will learn more about this very important concept and how you can use it to call built-in applications such as the Browser, Phone, Maps, and more.

UNDERSTANDING ACTIVITIES

This chapter begins by looking at how to create an activity. To create an activity, you create a Java class that extends the `Activity` base class:

```
package net.learn2develop.Activity101;

import android.app.Activity;
import android.os.Bundle;

public class Activity101Activity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Your activity class loads its UI component using the XML file defined in your `res/layout` folder. In this example, you would load the UI from the `main.xml` file:

```
        setContentView(R.layout.main);
```

Every activity you have in your application must be declared in your `AndroidManifest.xml` file, like this:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.Activity101"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".Activity101Activity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

The `Activity` base class defines a series of events that govern the life cycle of an activity. The `Activity` class defines the following events:

- `onCreate()` — Called when the activity is first created
- `onStart()` — Called when the activity becomes visible to the user
- `onResume()` — Called when the activity starts interacting with the user

- `onPause()` — Called when the current activity is being paused and the previous activity is being resumed
- `onStop()` — Called when the activity is no longer visible to the user
- `onDestroy()` — Called before the activity is destroyed by the system (either manually or by the system to conserve memory)
- `onRestart()` — Called when the activity has been stopped and is restarting again

By default, the activity created for you contains the `onCreate()` event. Within this event handler is the code that helps to display the UI elements of your screen.

Figure 2-1 shows the life cycle of an activity and the various stages it goes through — from when the activity is started until it ends.

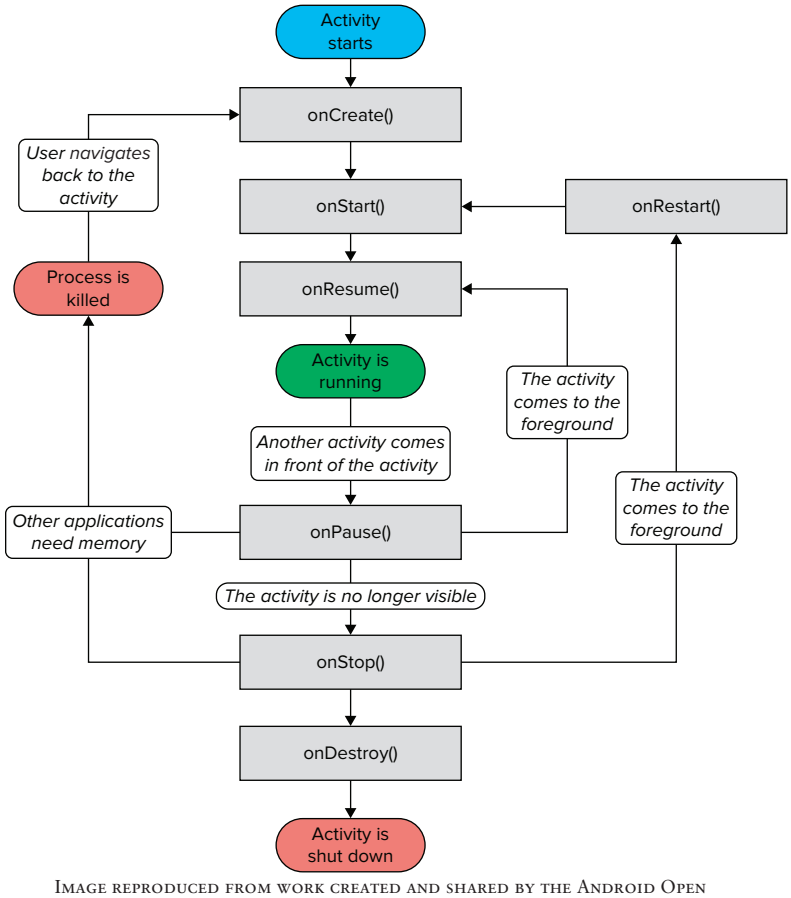


FIGURE 2-1

The best way to understand the various stages of an activity is to create a new project, implement the various events, and then subject the activity to various user interactions.

TRY IT OUT Understanding the Life Cycle of an Activity

codefile Activity101.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **Activity101**.
2. In the `Activity101Activity.java` file, add the following statements in bold:

```
package net.learn2develop.Activity101;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class Activity101Activity extends Activity {
    String tag = "Lifecycle";

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Log.d(tag, "In the onCreate() event");
    }

    public void onStart()
    {
        super.onStart();
        Log.d(tag, "In the onStart() event");
    }

    public void onRestart()
    {
        super.onRestart();
        Log.d(tag, "In the onRestart() event");
    }

    public void onResume()
    {
        super.onResume();
        Log.d(tag, "In the onResume() event");
    }

    public void onPause()
    {
        super.onPause();
        Log.d(tag, "In the onPause() event");
    }

    public void onStop()
```

```
    {
        super.onStop();
        Log.d(tag, "In the onStop() event");
    }

    public void onDestroy()
    {
        super.onDestroy();
        Log.d(tag, "In the onDestroy() event");
    }
}
```

3. Press F11 to debug the application on the Android emulator.

4. When the activity is first loaded, you should see something very similar to the following in the LogCat window (click the Debug perspective; see also Figure 2-2):

```
11-16 06:25:59.396: D/Lifecycle(559): In the onCreate() event
11-16 06:25:59.396: D/Lifecycle(559): In the onStart() event
11-16 06:25:59.396: D/Lifecycle(559): In the onResume() event
```

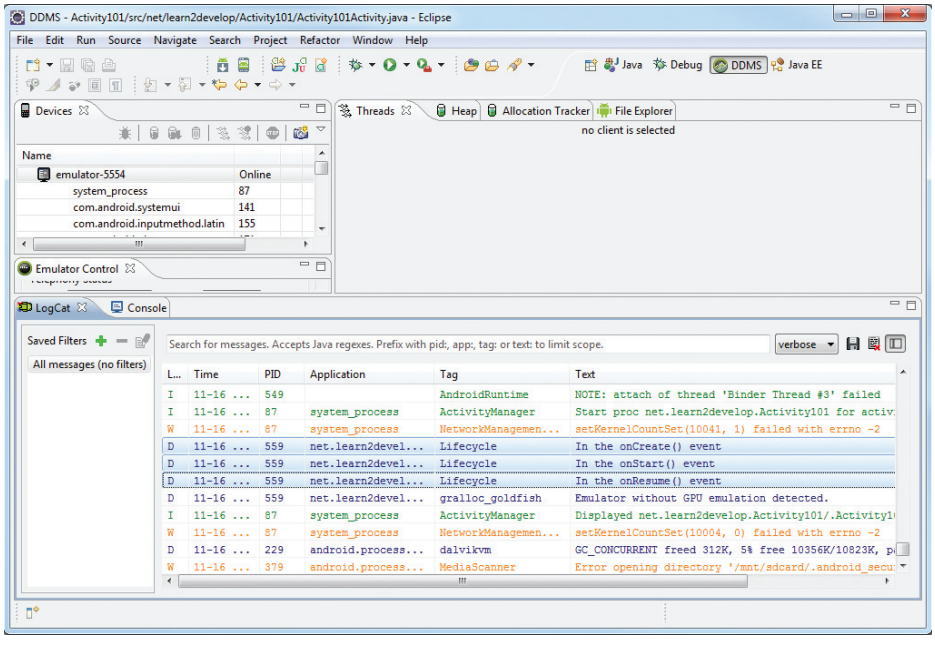


FIGURE 2-2

5. If you click the Back button on the Android emulator, the following is printed:

```
11-16 06:29:26.665: D/Lifecycle(559): In the onPause() event
11-16 06:29:28.465: D/Lifecycle(559): In the onStop() event
11-16 06:29:28.465: D/Lifecycle(559): In the onDestroy() event
```

6. Click the Home button and hold it there. Click the `Activities` icon and observe the following:

```
11-16 06:31:08.905: D/Lifecycle(559): In the onCreate() event
11-16 06:31:08.905: D/Lifecycle(559): In the onStart() event
11-16 06:31:08.925: D/Lifecycle(559): In the onResume() event
```

7. Click the Phone button on the Android emulator so that the activity is pushed to the background. Observe the output in the LogCat window:

```
11-16 06:32:00.585: D/Lifecycle(559): In the onPause() event
11-16 06:32:05.015: D/Lifecycle(559): In the onStop() event
```

8. Notice that the `onDestroy()` event is not called, indicating that the activity is still in memory. Exit the phone dialer by clicking the Back button. The activity is now visible again. Observe the output in the LogCat window:

```
11-16 06:32:50.515: D/Lifecycle(559): In the onRestart() event
11-16 06:32:50.515: D/Lifecycle(559): In the onStart() event
11-16 06:32:50.515: D/Lifecycle(559): In the onResume() event
```

The `onRestart()` event is now fired, followed by the `onStart()` and `onResume()` methods.

How It Works

As you can see from this simple example, an activity is destroyed when you click the Back button. This is crucial to know, as whatever state the activity is currently in will be lost; hence, you need to write additional code in your activity to preserve its state when it is destroyed (Chapter 3 shows you how). At this point, note that the `onPause()` method is called in both scenarios — when an activity is sent to the background, as well as when it is killed when the user presses the Back button.

When an activity is started, the `onStart()` and `onResume()`methods are always called, regardless of whether the activity is restored from the background or newly created. When an activity is created for the first time, the `onCreate()` method is called.

From the preceding example, you can derive the following guidelines:

- Use the `onCreate()` method to create and instantiate the objects that you will be using in your application.
- Use the `onResume()` method to start any services or code that needs to run while your activity is in the foreground.
- Use the `onPause()` method to stop any services or code that does not need to run when your activity is not in the foreground.
- Use the `onDestroy()` method to free up resources before your activity is destroyed.

NOTE *Even if an application has only one activity and the activity is killed, the application will still be running in memory.*

Applying Styles and Themes to an Activity

By default, an activity occupies the entire screen. However, you can apply a dialog theme to an activity so that it is displayed as a floating dialog. For example, you might want to customize your activity to display as a pop-up, warning users about some actions that they are going to perform. In this case, displaying the activity as a dialog is a good way to get their attention.

To apply a dialog theme to an activity, simply modify the `<Activity>` element in the `AndroidManifest.xml` file by adding the `android:theme` attribute:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.Activity101"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@android:style/Theme.Dialog">
        <activity
            android:label="@string/app_name"
            android:name=".Activity101Activity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

This will make the activity appear as a dialog, as shown in Figure 2-3.

FIGURE 2-3

Hiding the Activity Title

You can also hide the title of an activity if desired (such as when you just want to display a status update to the user). To do so, use the `requestWindowFeature()` method and pass it the `Window.FEATURE_NO_TITLE` constant, like this:

```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.Window;

public class Activity101Activity extends Activity {
```

```
    String tag = "Lifecycle";

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //---hides the title bar---
        requestWindowFeature(Window.FEATURE_NO_TITLE);

        setContentView(R.layout.main);
        Log.d(tag, "In the onCreate() event");
    }
}
```

This will hide the title bar, as shown in Figure 2-4.

FIGURE 2-4

Displaying a Dialog Window

There are times when you need to display a dialog window to get a confirmation from the user. In this case, you can override the `onCreateDialog()` protected method defined in the `Activity` base class to display a dialog window. The following Try It Out shows you how.

TRY IT OUT Displaying a Dialog Window Using an Activity

codefile Dialog.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **Dialog**.
2. Add the following statements in bold to the `main.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<Button
    android:id="@+id/btn_dialog"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click to display a dialog"
    android:onClick="onClick" />

</LinearLayout>
```

3. Add the following statements in bold to the `DialogActivity.java` file:

```
package net.learn2develop.Dialog;

import android.app.Activity;
```

```
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class DialogActivity extends Activity {
    CharSequence[] items = { "Google", "Apple", "Microsoft" };
    boolean[] itemsChecked = new boolean [items.length];

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClick(View v) {
        showDialog(0);
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
        case 0:
            return new AlertDialog.Builder(this)
            .setIcon(R.drawable.ic_launcher)
            .setTitle("This is a dialog with some simple text...")
            .setPositiveButton("OK",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton)
                    {
                        Toast.makeText(getBaseContext(),
                                "OK clicked!", Toast.LENGTH_SHORT).show();
                    }
                }
            )
            .setNegativeButton("Cancel",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton)
                    {
                        Toast.makeText(getBaseContext(),
                                "Cancel clicked!", Toast.LENGTH_SHORT).show();
                    }
                }
            )
            .setMultiChoiceItems(items, itemsChecked,
                new DialogInterface.OnMultiChoiceClickListener() {
                    public void onClick(DialogInterface dialog,
                    int which, boolean isChecked) {
                        Toast.makeText(getBaseContext(),
                            items[which] + (isChecked ? " checked!":" unchecked!"),
                            Toast.LENGTH_SHORT).show();
```

```
                    }
                }
            ).create();


        }
        return null;
    }

}
```

4. Press F11 to debug the application on the Android emulator. Click the button to display the dialog (see Figure 2-5). Checking the various checkboxes will cause the `Toast` class to display the text of the item checked/unchecked. To dismiss the dialog, click the OK or Cancel button.

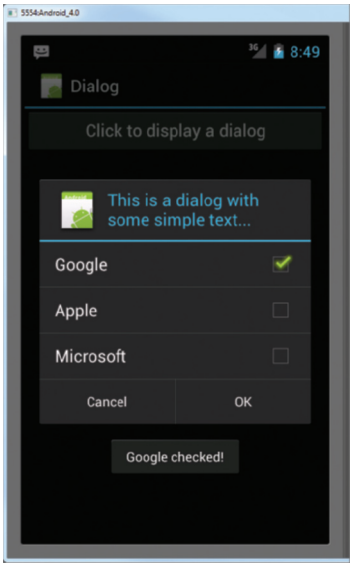


FIGURE 2-5

How It Works

To display a dialog, you first implement the `onCreateDialog()` method in the `Activity` class:

```
@Override
protected Dialog onCreateDialog(int id) {
    //...
}
```

This method is called when you call the `showDialog()` method:

```
public void onClick(View v) {
    showDialog(0);
}
```

The `onCreateDialog()` method is a callback for creating dialogs that are managed by the activity. When you call the `showDialog()` method, this callback will be invoked. The `showDialog()` method accepts an integer argument identifying a particular dialog to display. In this case, we used a `switch` statement to identify the different types of dialogs to create, although the current example creates only one type of dialog. Subsequent Try It Out exercises will extend this example to create different types of dialogs.

To create a dialog, you use the `AlertDialog` class's `Builder` constructor. You set the various properties, such as icon, title, and buttons, as well as checkboxes:

```
@Override
protected Dialog onCreateDialog(int id) {
    switch (id) {
    case 0:
        return new AlertDialog.Builder(this)
        .setIcon(R.drawable.ic_launcher)
        .setTitle("This is a dialog with some simple text...")
        .setPositiveButton("OK",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int whichButton)
```

```
                    {
                        Toast.makeText(getBaseContext(),
                                "OK clicked!", Toast.LENGTH_SHORT).show();
                    }
                }
            )
            .setNegativeButton("Cancel",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton)
                    {
                        Toast.makeText(getBaseContext(),
                                "Cancel clicked!", Toast.LENGTH_SHORT).show();
                    }
                }
            )
            .setMultiChoiceItems(items, itemsChecked,
                new DialogInterface.OnMultiChoiceClickListener() {
                    public void onClick(DialogInterface dialog,
                    int which, boolean isChecked) {
                        Toast.makeText(getBaseContext(),
                            items[which] + (isChecked ? " checked!":" unchecked!"),
                            Toast.LENGTH_SHORT).show();
                    }
                }
            ).create();


        }
        return null;
    }
```

The preceding code sets two buttons, OK and Cancel, using the `setPositiveButton()` and `setNegativeButton()` methods, respectively. You also set a list of checkboxes for users to choose via the `setMultiChoiceItems()` method. For the `setMultiChoiceItems()` method, you passed in two arrays: one for the list of items to display and another to contain the value of each item, to indicate if they are checked. When each item is checked, you use the `Toast` class to display a message indicating the item that was checked.

The preceding code for creating the dialog looks complicated, but it could easily be rewritten as follows:

```
package net.learn2develop.Dialog;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.AlertDialog.Builder;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class DialogActivity extends Activity {
```

```
    CharSequence[] items = { "Google", "Apple", "Microsoft" };
    boolean[] itemsChecked = new boolean [items.length];

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClick(View v) {
        showDialog(0);
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
        case 0:
            Builder builder = new AlertDialog.Builder(this);
            builder.setIcon(R.drawable.ic_launcher);
            builder.setTitle("This is a dialog with some simple text...");
            builder.setPositiveButton("OK",
                new DialogInterface.OnClickListener() {
                   public void onClick(DialogInterface dialog,    int whichButton) {
                        Toast.makeText(getBaseContext(),
                                "OK clicked!", Toast.LENGTH_SHORT).show();
                    }
                }
            );

            builder.setNegativeButton("Cancel",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton) {
                        Toast.makeText(getBaseContext(),
                            "Cancel clicked!", Toast.LENGTH_SHORT).show();
                    }
                }
            );

            builder.setMultiChoiceItems(items, itemsChecked,
                new DialogInterface.OnMultiChoiceClickListener() {
                    public void onClick(DialogInterface dialog,
                    int which, boolean isChecked) {
                        Toast.makeText(getBaseContext(),
                            items[which] + (isChecked ? " checked!":" unchecked!"),
                            Toast.LENGTH_SHORT).show();
                    }
                }
            );
            return builder.create();
        }
        return null;
    }
}
```

THE CONTEXT OBJECT

In Android, you often encounter the `Context` class and its instances. Instances of the `Context` class are often used to provide references to your application. For example, in the following code snippet, the first parameter of the `Toast` class takes in a `Context` object:

```
.setPositiveButton("OK",
    new DialogInterface.OnClickListener() {
       public void onClick(DialogInterface dialog, int whichButton)
        {
            Toast.makeText(getBaseContext(),
                    "OK clicked!", Toast.LENGTH_SHORT).show();
        }
    }
```

However, because the `Toast()` class is not used directly in the activity (it is used within the `AlertDialog` class), you need to return an instance of the `Context` class by using the `getBaseContext()` method.

You also encounter the `Context` class when creating a view dynamically in an activity. For example, you may want to dynamically create a `TextView` from code. To do so, you instantiate the `TextView` class, like this:

```
TextView tv = new TextView(this);
```

The constructor for the `TextView` class takes a `Context` object; and because the `Activity` class is a subclass of `Context`, you can use the `this` keyword to represent the `Context` object.

Displaying a Progress Dialog

One common UI feature in an Android device is the "Please wait" dialog that you typically see when an application is performing a long-running task. For example, the application may be logging in to a server before the user is allowed to use it, or it may be doing a calculation before displaying the result to the user. In such cases, it is helpful to display a dialog, known as a *progress dialog*, so that the user is kept in the loop.

The following Try It Out demonstrates how to display such a dialog.

TRY IT OUT Displaying a Progress (Please Wait) Dialog

1. Using the same project created in the previous section, add the following statements in bold to the `main.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<Button
    android:id="@+id/btn_dialog"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click to display a dialog"
    android:onClick="onClick" />

<Button
    android:id="@+id/btn_dialog2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click to display a progress dialog"
    android:onClick="onClick2" />

</LinearLayout>
```

2. Add the following statements in bold to the `DialogActivity.java` file:

```
package net.learn2develop.Dialog;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.AlertDialog.Builder;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class DialogActivity extends Activity {
    CharSequence[] items = { "Google", "Apple", "Microsoft" };
    boolean[] itemsChecked = new boolean [items.length];

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClick(View v) {
        showDialog(0);
    }

    public void onClick2(View v) {
        //---show the dialog---
        final ProgressDialog dialog = ProgressDialog.show(
            this, "Doing something", "Please wait...", true);
        new Thread(new Runnable(){
```

```
            public void run(){
                try {
                    //---simulate doing something lengthy---
                    Thread.sleep(5000);
                    //---dismiss the dialog---
                    dialog.dismiss();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }).start();
    }

    @Override
    protected Dialog onCreateDialog(int id) { ... }

}
```

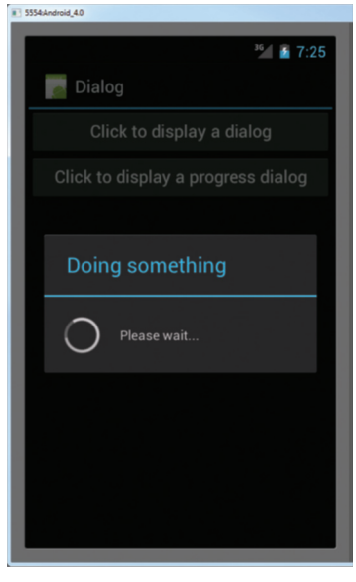


FIGURE 2-6

3. Press F11 to debug the application on the Android emulator. Clicking the second button will display the progress dialog, as shown in Figure 2-6. It will go away after five seconds.

How It Works

Basically, to create a progress dialog, you created an instance of the `ProgressDialog` class and called its `show()` method:

```
//---show the dialog---
final ProgressDialog dialog = ProgressDialog.show(
    this, "Doing something", "Please wait...", true);
```

This displays the progress dialog that you have just seen. Because this is a modal dialog, it will block the UI until it is dismissed. To perform a long-running task in the background, you created a `Thread` using a `Runnable` block (you will learn more about threading in Chapter 11). The code that you placed inside the `run()` method will be executed in a separate thread, and in this case you simulated it performing something for five seconds by inserting a delay using the `sleep()` method:

```
new Thread(new Runnable(){
    public void run(){
        try {
            //---simulate doing something lengthy---
            Thread.sleep(5000);
            //---dismiss the dialog---
            dialog.dismiss();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}).start();
```

After the five seconds elapse, you dismiss the dialog by calling the `dismss()` method.

Displaying a More Sophisticated Progress Dialog

Besides the generic "please wait" dialog that you created in the previous section, you can also create a dialog that displays the progress of an operation, such as the status of a download.

The following Try It Out shows you how to display a specialized progress dialog.

TRY IT OUT Displaying the Progress of an Operation

1. Using the same project created in the previous section, add the following lines in bold to the `main.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<Button
    android:id="@+id/btn_dialog"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click to display a dialog"
    android:onClick="onClick" />

<Button
    android:id="@+id/btn_dialog2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click to display a progress dialog"
    android:onClick="onClick2" />

<Button
    android:id="@+id/btn_dialog3"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click to display a detailed progress dialog"
    android:onClick="onClick3" />

</LinearLayout>
```

2. Add the following statements in bold to the `DialogActivity.java` file:

```
package net.learn2develop.Dialog;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.AlertDialog.Builder;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.Toast;

public class DialogActivity extends Activity {
    CharSequence[] items = { "Google", "Apple", "Microsoft" };
    boolean[] itemsChecked = new boolean [items.length];

    ProgressDialog progressDialog;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) { ... }

    public void onClick(View v) { ... }

    public void onClick2(View v) { ... }

    public void onClick3(View v) {
        showDialog(1);
        progressDialog.setProgress(0);

        new Thread(new Runnable(){
            public void run(){
                for (int i=1; i<=15; i++) {
                    try {
                        //---simulate doing something lengthy---
                        Thread.sleep(1000);
                        //---update the dialog---
                        progressDialog.incrementProgressBy((int)(100/15));
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                progressDialog.dismiss();
            }
        }).start();
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
        case 0:
            return new AlertDialog.Builder(this)
                //...
            ).create();

        case 1:
            progressDialog = new ProgressDialog(this);
            progressDialog.setIcon(R.drawable.ic_launcher);
            progressDialog.setTitle("Downloading files...");
            progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
            progressDialog.setButton(DialogInterface.BUTTON_POSITIVE, "OK",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog,
                    int whichButton)
```

```
                {
                    Toast.makeText(getBaseContext(),
                            "OK clicked!", Toast.LENGTH_SHORT).show();
                }
            });
            progressDialog.setButton(DialogInterface.BUTTON_NEGATIVE, "Cancel",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog,
                        int whichButton)
                    {
                        Toast.makeText(getBaseContext(),
                                "Cancel clicked!", Toast.LENGTH_SHORT).show();
                    }
                });
                return progressDialog;
            }

            return null;
        }
    }
```

3. Press F11 to debug the application on the Android emulator. Click the third button to display the progress dialog (see Figure 2-7). Note that the progress bar will count up to 100%.

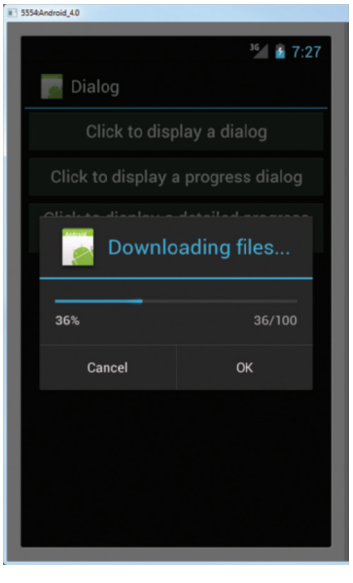


FIGURE 2-7

How It Works

To create a dialog that shows the progress of an operation, you first create an instance of the `ProgressDialog` class and set its various properties, such as icon, title, and style:

```
progressDialog = new ProgressDialog(this);
progressDialog.setIcon(R.drawable.ic_launcher);
progressDialog.setTitle("Downloading files...");
progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
```

You then set the two buttons that you want to display inside the progress dialog:

```
progressDialog.setButton(DialogInterface.BUTTON_POSITIVE, "OK",
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog,
        int whichButton)
        {
            Toast.makeText(getBaseContext(),
                    "OK clicked!", Toast.LENGTH_SHORT).show();
        }
});
progressDialog.setButton(DialogInterface.BUTTON_NEGATIVE, "Cancel",
    new DialogInterface.OnClickListener() {
```

```
                public void onClick(DialogInterface dialog,
                    int whichButton)
                {
                    Toast.makeText(getBaseContext(),
                            "Cancel clicked!", Toast.LENGTH_SHORT).show();
                }
            });
            return progressDialog;
```

The preceding code causes a progress dialog to appear (see Figure 2-8).

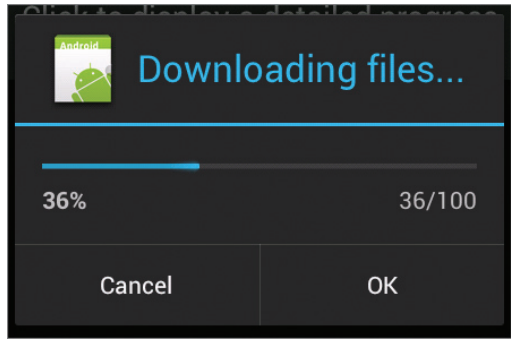


FIGURE 2-8

To display the progress status in the progress dialog, you can use a `Thread` object to run a `Runnable` block of code:

```
progressDialog.setProgress(0);

new Thread(new Runnable(){
    public void run(){
        for (int i=1; i<=15; i++) {
            try {
                //---simulate doing something lengthy---
                Thread.sleep(1000);
                //---update the dialog---
                progressDialog.incrementProgressBy((int)(100/15));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        progressDialog.dismiss();
    }
}).start();
```

In this case, you want to count from 1 to 15, with a delay of one second between each number. The `incrementProgressBy()` method increments the counter in the progress dialog. When the progress dialog reaches 100%, it is dismissed.

LINKING ACTIVITIES USING INTENTS

An Android application can contain zero or more activities. When your application has more than one activity, you often need to navigate from one to another. In Android, you navigate between activities through what is known as an *intent*.

The best way to understand this very important but somewhat abstract concept in Android is to experience it firsthand and see what it helps you to achieve. The following Try It Out shows how to add another activity to an existing project and then navigate between the two activities.

TRY IT OUT Linking Activities with Intents

codefile UsingIntent.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **UsingIntent**.
2. Right-click on the package name under the `src` folder and select New ➪ Class (see Figure 2-9).

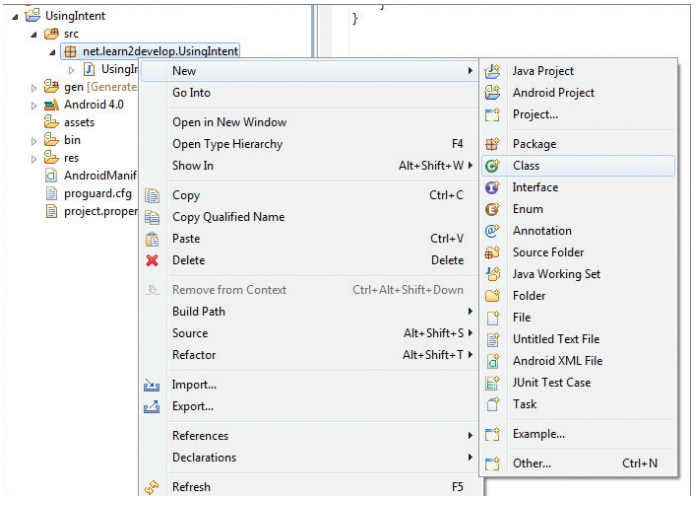


FIGURE 2-9

3. Name the new class **SecondActivity** and click Finish.
4. Add the following statements in bold to the `AndroidManifest.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.UsingIntent"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".UsingIntentActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:label="Second Activity"
            android:name=".SecondActivity" >
```

```
            <intent-filter >
                <action android:name="net.learn2develop.SecondActivity" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

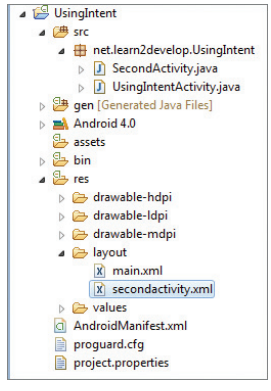


FIGURE 2-10

5. Make a copy of the `main.xml` file (in the `res/layout` folder) by right-clicking on it and selecting `Copy`. Then, right-click on the `res/layout` folder and select `Paste`. Name the file `secondactivity.xml`. The `res/layout` folder will now contain the `secondactivity.xml` file (see Figure 2-10).

6. Modify the `secondactivity.xml` file as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="This is the Second Activity!" />

</LinearLayout>
```

7. In the `SecondActivity.java` file, add the following statements in bold:

```
package net.learn2develop.UsingIntent;

import android.app.Activity;
import android.os.Bundle;

public class SecondActivity extends Activity{
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.secondactivity);
    }
}
```

8. Add the following lines in bold to the `main.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<Button
```

```
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Display second activity"
    android:onClick="onClick"/>

</LinearLayout>
```

9. Modify the `UsingIntentActivity.java` file as shown in bold:

```
package net.learn2develop.UsingIntent;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class UsingIntentActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClick(View view) {
        startActivity(new Intent("net.learn2develop.SecondActivity"));
    }
}
```

10. Press F11 to debug the application on the Android emulator. When the first activity is loaded, click the button and the second activity will now be loaded (see Figure 2-11).

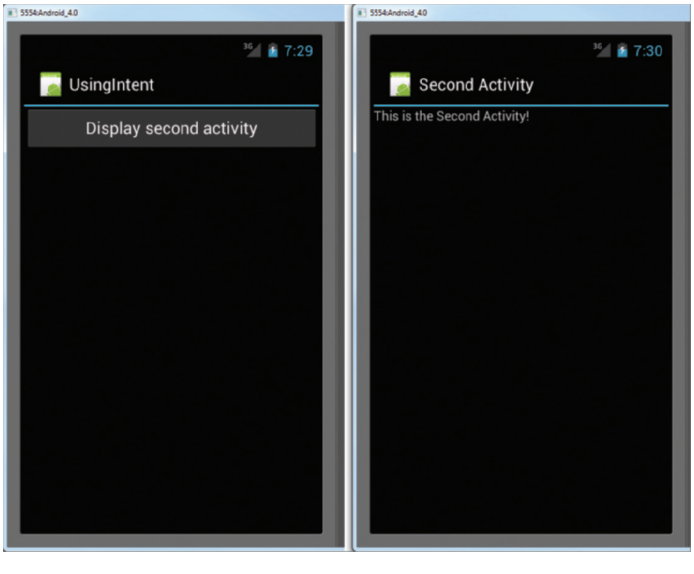


FIGURE 2-11

How It Works

As you have learned, an activity is made up of a UI component (for example, `main.xml`) and a class component (for example, `UsingIntentActivity.java`). Hence, if you want to add another activity to a project, you need to create these two components.

In the `AndroidManifest.xml` file, specifically you have added the following:

```
<activity
    android:label=" Second Activity"
    android:name=".SecondActivity" >
    <intent-filter >
        <action android:name="net.learn2develop.SecondActivity" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Here, you have added a new activity to the application. Note the following:

- The name (class) of the new activity added is `SecondActivity`.
- The label for the new activity is named `Second Activity`.
- The intent filter name for the new activity is `net.learn2develop.SecondActivity`. Other activities that wish to call this activity will invoke it via this name. Ideally, you should use the reverse domain name of your company as the intent filter name in order to reduce the chances of another application having the same intent filter name. (The next section discusses what happens when two or more activities have the same intent filter.)
- The category for the intent filter is `android.intent.category.DEFAULT`. You need to add this to the intent filter so that this activity can be started by another activity using the `startActivity()` method (more on this shortly).

When the button is clicked, you use the `startActivity()` method to display `SecondActivity` by creating an instance of the `Intent` class and passing it the intent filter name of `SecondActivity` (which is `net.learn2develop.SecondActivity`):

```
public void onClick(View view) {
    startActivity(new Intent("net.learn2develop.SecondActivity"));
}
```

Activities in Android can be invoked by any application running on the device. For example, you can create a new Android project and then display `SecondActivity` by using its `net.learn2develop.SecondActivity` intent filter. This is one of the fundamental concepts in Android that enables an application to invoke another easily.

If the activity that you want to invoke is defined within the same project, you can rewrite the preceding statement like this:

```
startActivity(new Intent(this, SecondActivity.class));
```

However, this approach is applicable only when the activity you want to display is within the same project as the current activity.

Resolving Intent Filter Collision

In the previous section, you learned that the `<intent-filter>` element defines how your activity can be invoked by another activity. What happens if another activity (in either the same or a separate application) has the same filter name? For example, suppose your application has another activity named `Activity3`, with the following entry in the `AndroidManifest.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.UsingIntent"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".UsingIntentActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:label="Second Activity"
            android:name=".SecondActivity" >
            <intent-filter >
                <action android:name="net.learn2develop.SecondActivity" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>

        <activity
            android:label="Third Activity"
            android:name=".ThirdActivity" >
            <intent-filter >
                <action android:name="net.learn2develop.SecondActivity" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>

    </application>

</manifest>
```

If you call the `startActivity()` method with the following intent, then the Android OS will display a selection of activities, as shown in Figure 2-12:

```
startActivity(new Intent("net.learn2develop.SecondActivity"));
```

If you check the "Use by default for this action" item and then select an activity, then the next time the intent "`net.learn2develop.SecondActivity`" is called again, it will launch the previous activity that you have selected.

To clear this default, go to the Settings application in Android and select Apps ➪ Manage applications, and then select the application name (see Figure 2-13). When the details of the application are shown, scroll down to the bottom and click the Clear defaults button.

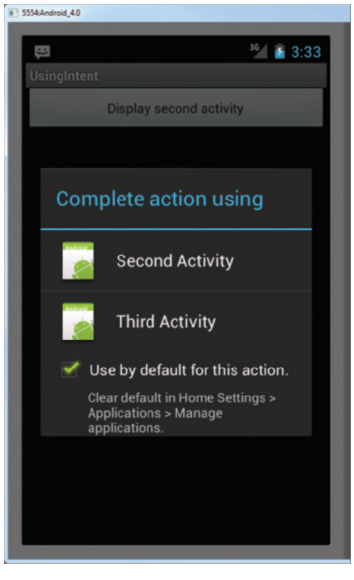


FIGURE 2-12

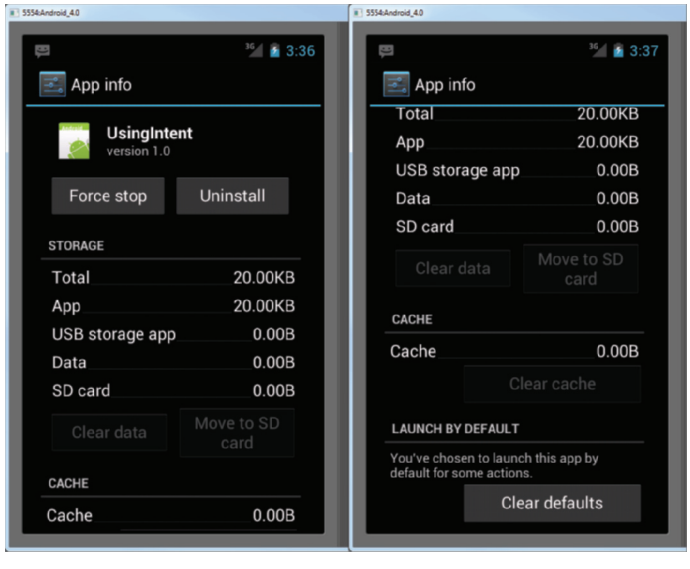


FIGURE 2-13

Returning Results from an Intent

The `startActivity()` method invokes another activity but does not return a result to the current activity. For example, you may have an activity that prompts the user for user name and password. The information entered by the user in that activity needs to be passed back to the calling activity for further processing. If you need to pass data back from an activity, you should instead use the `startActivityForResult()` method. The following Try It Out demonstrates this.

TRY IT OUT Obtaining a Result from an Activity

1. Using the same project from the previous section, add the following statements in bold to the `secondactivity.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```

```
    android:orientation="vertical" >

<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="This is the Second Activity!" />

<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Please enter your name" />

<EditText
    android:id="@+id/txt_username"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />

<Button
    android:id="@+id/btn_OK"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="OK"
    android:onClick="onClick"/>

</LinearLayout>
```

2. Add the following statements in bold to `SecondActivity.java`:

```
package net.learn2develop.UsingIntent;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class SecondActivity extends Activity{
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.secondactivity);
    }

    public void onClick(View view) {
        Intent data = new Intent();

        //---get the EditText view---
        EditText txt_username =
            (EditText) findViewById(R.id.txt_username);

        //---set the data to pass back---
        data.setData(Uri.parse(
```

```
            txt_username.getText().toString()));
        setResult(RESULT_OK, data);

        //---closes the activity---
        finish();
    }
}
```

3. Add the following statements in bold to the `UsingIntentActivity.java` file:

```
package net.learn2develop.UsingIntent;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class UsingIntentActivity extends Activity {
    int request_Code = 1;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClick(View view) {
        //startActivity(new Intent("net.learn2develop.SecondActivity"));
        //or
        //startActivity(new Intent(this, SecondActivity.class));
        startActivityForResult(new Intent(
            "net.learn2develop.SecondActivity"),
            request_Code);
    }

    public void onActivityResult(int requestCode, int resultCode, Intent data)
    {
        if (requestCode == request_Code) {
            if (resultCode == RESULT_OK) {
                Toast.makeText(this,data.getData().toString(),
                    Toast.LENGTH_SHORT).show();
            }
        }
    }
}
```

4. Press F11 to debug the application on the Android emulator. When the first activity is loaded, click the button. `SecondActivity` will now be loaded. Enter your name (see Figure 2-14) and click the OK button. The first activity will display the name you have entered using the `Toast` class.

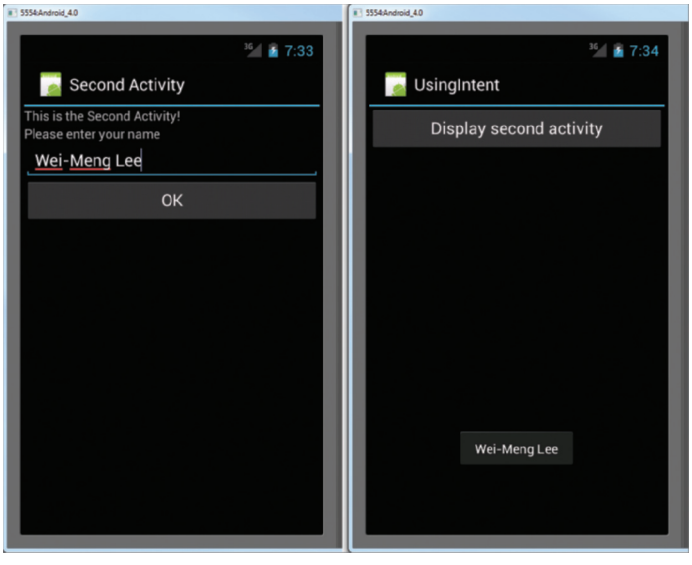


FIGURE 2-14

How It Works

To call an activity and wait for a result to be returned from it, you need to use the `startActivityForResult()` method, like this:

```
startActivityForResult(new Intent(
    "net.learn2develop.SecondActivity"),
    request_Code);
```

In addition to passing in an `Intent` object, you need to pass in a request code as well. The request code is simply an integer value that identifies an activity you are calling. This is needed because when an activity returns a value, you must have a way to identify it. For example, you may be calling multiple activities at the same time, and some activities may not return immediately (for example, waiting for a reply from a server). When an activity returns, you need this request code to determine which activity is actually returned.

NOTE *If the request code is set to -1, then calling it using the* `startActivityForResult()` *method is equivalent to calling it using the* `startActivity()` *method. That is, no result will be returned.*

In order for an activity to return a value to the calling activity, you use an `Intent` object to send data back via the `setData()` method:

```
Intent data = new Intent();

//---get the EditText view---
```

```
EditText txt_username =
    (EditText) findViewById(R.id.txt_username);

//---set the data to pass back---
data.setData(Uri.parse(
    txt_username.getText().toString()));
setResult(RESULT_OK, data);

//---closes the activity---
finish();
```

The `setResult()` method sets a result code (either `RESULT_OK` or `RESULT_CANCELLED`) and the data (an `Intent` object) to be returned back to the calling activity. The `finish()` method closes the activity and returns control back to the calling activity.

In the calling activity, you need to implement the `onActivityResult()` method, which is called whenever an activity returns:

```
public void onActivityResult(int requestCode, int resultCode,
Intent data)
{
    if (requestCode == request_Code) {
        if (resultCode == RESULT_OK) {
            Toast.makeText(this,data.getData().toString(),
                Toast.LENGTH_SHORT).show();
        }
    }
}
```

Here, you check for the appropriate request and result codes and display the result that is returned. The returned result is passed in via the `data` argument; and you obtain its details through the `getData()` method.

Passing Data Using an Intent Object

Besides returning data from an activity, it is also common to pass data to an activity. For example, in the previous example you may want to set some default text in the `EditText` view before the activity is displayed. In this case, you can use the `Intent` object to pass the data to the target activity.

The following Try It Out shows you the various ways in which you can pass data between activities.

TRY IT OUT Passing Data to the Target Activity

1. Using Eclipse, create a new Android project and name it **PassingData**.
2. Add the following statements in bold to the `main.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```

```
    android:orientation="vertical" >

<Button
    android:id="@+id/btn_SecondActivity"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click to go to Second Activity"
    android:onClick="onClick"/>

</LinearLayout>
```

3. Add a new XML file to the `res/layout` folder and name it `secondactivity.xml`. Populate it as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Welcome to Second Activity" />

<Button
    android:id="@+id/btn_MainActivity"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Click to return to main activity"
    android:onClick="onClick"/>

</LinearLayout>
```

4. Add a new Class file to the package and name it `SecondActivity`. Populate the `SecondActivity.java` file as follows:

```
package net.learn2develop.PassingData;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class SecondActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.secondactivity);

        //---get the data passed in using getStringExtra()---
        Toast.makeText(this,getIntent().getStringExtra("str1"),
                Toast.LENGTH_SHORT).show();

        //---get the data passed in using getIntExtra()---
```

```
        Toast.makeText(this,Integer.toString(
                getIntent().getIntExtra("age1", 0)),
                Toast.LENGTH_SHORT).show();

        //---get the Bundle object passed in---
        Bundle bundle = getIntent().getExtras();

        //---get the data using the getString()---
        Toast.makeText(this, bundle.getString("str2"),
                Toast.LENGTH_SHORT).show();

        //---get the data using the getInt() method---
        Toast.makeText(this,Integer.toString(bundle.getInt("age2")),
                Toast.LENGTH_SHORT).show();
    }

    public void onClick(View view) {
        //---use an Intent object to return data---
        Intent i = new Intent();

        //---use the putExtra() method to return some
        // value---
        i.putExtra("age3", 45);

        //---use the setData() method to return some value---
        i.setData(Uri.parse(
                "Something passed back to main activity"));

        //---set the result with OK and the Intent object---
        setResult(RESULT_OK, i);

        //---destroy the current activity---
        finish();
    }
}
```

5. Add the following statements in bold to the `AndroidManifest.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.PassingData"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />

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

```
        <activity
            android:label="Second Activity"
            android:name=".SecondActivity" >
            <intent-filter >
                <action android:name="net.learn2develop.PassingDataSecondActivity" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

6. Add the following statements in bold to the `PassingDataActivity.java` file:

```
package net.learn2develop.PassingData;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PassingDataActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClick(View view) {
        Intent i = new
                Intent("net.learn2develop.PassingDataSecondActivity");
        //---use putExtra() to add new name/value pairs---
        i.putExtra("str1", "This is a string");
        i.putExtra("age1", 25);

        //---use a Bundle object to add new name/values
        // pairs---
        Bundle extras = new Bundle();
        extras.putString("str2", "This is another string");
        extras.putInt("age2", 35);

        //---attach the Bundle object to the Intent object---
        i.putExtras(extras);

        //---start the activity to get a result back---
        startActivityForResult(i, 1);
    }

    public void onActivityResult(int requestCode,
    int resultCode, Intent data)
    {
        //---check if the request code is 1---
        if (requestCode == 1) {

            //---if the result is OK---
```

```
            if (resultCode == RESULT_OK) {

                //---get the result using getIntExtra()---
                Toast.makeText(this, Integer.toString(
                    data.getIntExtra("age3", 0)),
                    Toast.LENGTH_SHORT).show();

                //---get the result using getData()---
                Toast.makeText(this, data.getData().toString(),
                    Toast.LENGTH_SHORT).show();
            }
        }
    }

}
```

7. Press F11 to debug the application on the Android emulator. Click the button on each activity and observe the values displayed.

How It Works

While this application is not visually exciting, it does illustrate some important ways to pass data between activities.

First, you can use the `putExtra()` method of an `Intent` object to add a name/value pair:

```
//---use putExtra() to add new name/value pairs---
i.putExtra("str1", "This is a string");
i.putExtra("age1", 25);
```

The preceding statements add two name/value pairs to the `Intent` object: one of type `string` and one of type `integer`.

Besides using the `putExtra()` method, you can also create a `Bundle` object and then attach it using the `putExtras()` method. Think of a `Bundle` object as a dictionary object — it contains a set of name/value pairs. The following statements create a `Bundle` object and then add two name/value pairs to it. It is then attached to the `Intent` object:

```
//---use a Bundle object to add new name/values pairs---
Bundle extras = new Bundle();
extras.putString("str2", "This is another string");
extras.putInt("age2", 35);

//---attach the Bundle object to the Intent object---
i.putExtras(extras);
```

On the second activity, to obtain the data sent using the `Intent` object, you first obtain the `Intent` object using the `getIntent()` method. Then, call its `getStringExtra()` method to get the string value set using the `putExtra()` method:

```
//---get the data passed in using getStringExtra()---
Toast.makeText(this,getIntent().getStringExtra("str1"),
        Toast.LENGTH_SHORT).show();
```

In this case, you have to call the appropriate method to extract the name/value pair based on the type of data set. For the integer value, use the `getIntExtra()` method (the second argument is the default value in case no value is stored in the specified name):

```
//---get the data passed in using getIntExtra()---
Toast.makeText(this,Integer.toString(
        getIntent().getIntExtra("age1", 0)),
        Toast.LENGTH_SHORT).show();
```

To retrieve the `Bundle` object, use the `getExtras()` method:

```
//---get the Bundle object passed in---
Bundle bundle = getIntent().getExtras();
```

To get the individual name/value pairs, use the appropriate method. For the string value, use the `getString()` method:

```
//---get the data using the getString()---
Toast.makeText(this, bundle.getString("str2"),
        Toast.LENGTH_SHORT).show();
```

Likewise, use the `getInt()` method to retrieve an integer value:

```
//---get the data using the getInt() method---
Toast.makeText(this,Integer.toString(bundle.getInt("age2")),
        Toast.LENGTH_SHORT).show();
```

Another way to pass data to an activity is to use the `setData()` method (as used in the previous section), like this:

```
//---use the setData() method to return some value---
i.setData(Uri.parse(
        "Something passed back to main activity"));
```

Usually, you use the `setData()` method to set the data on which an `Intent` object is going to operate (such as passing a URL to an `Intent` object so that it can invoke a web browser to view a web page; see the section "Calling Built-In Applications Using Intents" later in this chapter for more examples).

To retrieve the data set using the `setData()` method, use the `getData()` method (in this example `data` is an `Intent` object):

```
//---get the result using getData()---
Toast.makeText(this, data.getData().toString(),
    Toast.LENGTH_SHORT).show();
```

FRAGMENTS

In the previous section you learned what an activity is and how to use it. In a small-screen device (such as a smartphone), an activity typically fills the entire screen, displaying the various views that make up the user interface of an application. The activity is essentially a container for views. However, when an activity is displayed in a large-screen device, such as on a tablet, it is somewhat out of place. Because the screen is much bigger, all the views in an activity must be arranged to make full use of the increased space, resulting in complex changes to the view hierarchy. A better approach is to have "mini-activities," each containing its own set of views. During runtime, an activity can contain one or more of these mini-activities, depending on the screen orientation in which the device is held. In Android 3.0 and later, these mini-activities are known as *fragments*.

Think of a fragment as another form of activity. You create fragments to contain views, just like activities. Fragments are always embedded in an activity. For example, Figure 2-15 shows two fragments. Fragment 1 might contain a `ListView` showing a list of book titles. Fragment 2 might contain some `TextView`s and `ImageView`s showing some text and images.

Now imagine the application is running on an Android tablet in portrait mode (or on an Android smartphone). In this case, Fragment 1 may be embedded in one activity, while Fragment 2 may be embedded in another activity (see Figure 2-16). When users select an item in the list in Fragment 1, Activity 2 will be started.

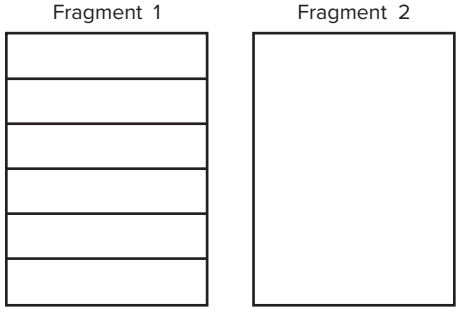


FIGURE 2-15

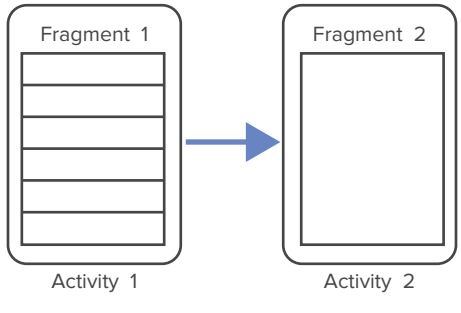


FIGURE 2-16

If the application is now displayed in a tablet in landscape mode, both fragments can be embedded within a single activity, as shown in Figure 2-17.

From this discussion, it becomes apparent that fragments present a versatile way in which you can create the user interface of an Android application. Fragments form the atomic unit of your user interface, and they can be dynamically added (or removed) to activities in order to create the best user experience possible for the target device.

The following Try It Out shows you the basics of working with fragments.

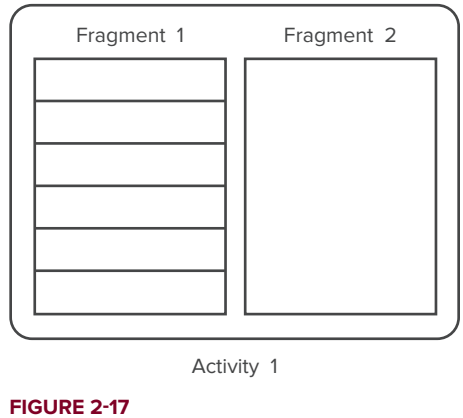


FIGURE 2-17

TRY IT OUT Using Fragments

codefile Fragments.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **Fragments**.
2. In the `res/layout` folder, add a new file and name it `fragment1.xml`. Populate it with the following:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#00FF00"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="This is fragment #1"
    android:textColor="#000000"
    android:textSize="25sp" />
</LinearLayout>
```

3. Also in the `res/layout` folder, add another new file and name it `fragment2.xml`. Populate it as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#FFFE00"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="This is fragment #2"
    android:textColor="#000000"
    android:textSize="25sp" />
</LinearLayout>
```

4. In `main.xml`, add the following code in bold:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >

    <fragment
        android:name="net.learn2develop.Fragments.Fragment1"
        android:id="@+id/fragment1"
```

```
            android:layout_weight="1"
            android:layout_width="0px"
            android:layout_height="match_parent" />
        <fragment
            android:name="net.learn2develop.Fragments.Fragment2"
            android:id="@+id/fragment2"
            android:layout_weight="1"
            android:layout_width="0px"
            android:layout_height="match_parent" />

</LinearLayout>
```

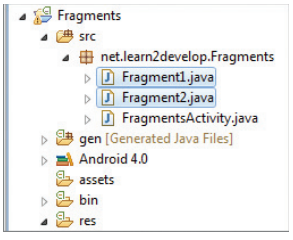


FIGURE 2-18

5. Under the `net.learn2develop.Fragments` package name, add two Java class files and name them `Fragment1.java` and `Fragment2.java` (see Figure 2-18).

6. Add the following code to `Fragment1.java`:

```
package net.learn2develop.Fragments;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class Fragment1 extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
    ViewGroup container, Bundle savedInstanceState) {
        //---Inflate the layout for this fragment---
        return inflater.inflate(
            R.layout.fragment1, container, false);
    }
}
```

7. Add the following code to `Fragment2.java`:

```
package net.learn2develop.Fragments;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class Fragment2 extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
    ViewGroup container, Bundle savedInstanceState) {
        //---Inflate the layout for this fragment---
        return inflater.inflate(
            R.layout.fragment2, container, false);
    }
}
```

8. Press F11 to debug the application on the Android emulator. Figure 2-19 shows the two fragments contained within the activity.

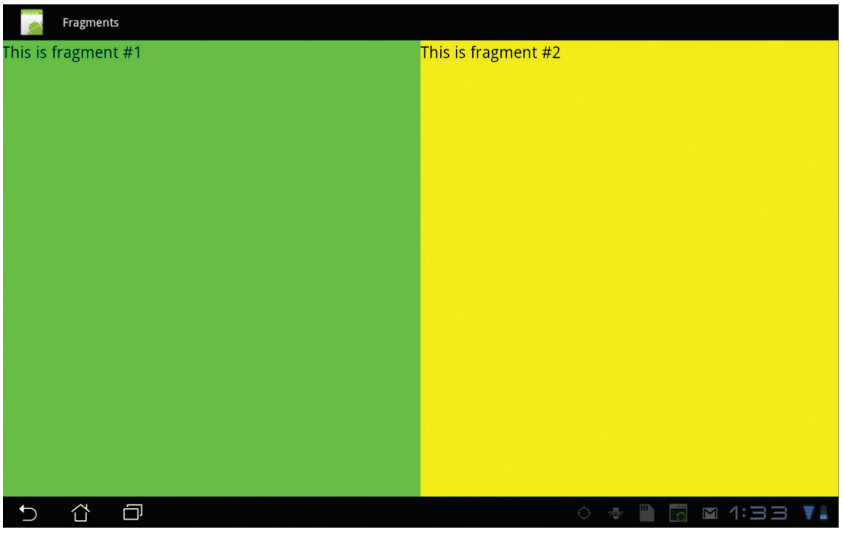


FIGURE 2-19

How It Works

A fragment behaves very much like an activity — it has a Java class and it loads its UI from an XML file. The XML file contains all the usual UI elements that you expect from an activity: `TextView`, `EditText`, `Button`, and so on. The Java class for a fragment needs to extend the `Fragment` base class:

```
public class Fragment1 extends Fragment {
}
```

NOTE *Besides the Fragment base class, a fragment can also extend a few other subclasses of the* `Fragment` *class, such as* `DialogFragment`, `ListFragment`, *and* `PreferenceFragment`. *Chapter 4 discusses these types of fragments in more detail.*

To draw the UI for a fragment, you override the `onCreateView()` method. This method needs to return a `View` object, like this:

```
public View onCreateView(LayoutInflater inflater,
ViewGroup container, Bundle savedInstanceState) {
    //---Inflate the layout for this fragment---
    return inflater.inflate(
```

```
                R.layout.fragment1, container, false);
    }
```

Here, you use a `LayoutInflater` object to inflate the UI from the specified XML file (`R.layout.fragment1` in this case). The `container` argument refers to the parent `ViewGroup`, which is the activity in which you are trying to embed the fragment. The `savedInstanceState` argument enables you to restore the fragment to its previously saved state.

To add a fragment to an activity, you use the `<fragment>` element:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >

    <fragment
        android:name="net.learn2develop.Fragments.Fragment1"
        android:id="@+id/fragment1"
        android:layout_weight="1"
        android:layout_width="0px"
        android:layout_height="match_parent" />
    <fragment
        android:name="net.learn2develop.Fragments.Fragment2"
        android:id="@+id/fragment2"
        android:layout_weight="1"
        android:layout_width="0px"
        android:layout_height="match_parent" />

</LinearLayout>
```

Note that each fragment needs a unique identifier. You can assign one via the `android:id` or `android:tag` attribute.

Adding Fragments Dynamically

While fragments enable you to compartmentalize your UI into various configurable parts, the real power of fragments is realized when you add them dynamically to activities during runtime. In the previous section, you saw how you can add fragments to an activity by modifying the XML file during design time. In reality, it is much more useful if you create fragments and add them to activities during runtime. This enables you to create a customizable user interface for your application. For example, if the application is running on a smartphone, you might fill an activity with a single fragment; if the application is running on a tablet, you might then fill the activity with two or more fragments, as the tablet has much more screen real estate compared to a smartphone.

The following Try It Out shows how fragments can be added programmatically to an activity during runtime.

TRY IT OUT Adding Fragments during Runtime

1. Using the same project created in the previous section, modify the `main.xml` file by commenting out the two `<fragment>` elements:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >

    <!--
    <fragment
        android:name="net.learn2develop.Fragments.Fragment1"
        android:id="@+id/fragment1"
        android:layout_weight="1"
        android:layout_width="0px"
        android:layout_height="match_parent" />
    <fragment
        android:name="net.learn2develop.Fragments.Fragment2"
        android:id="@+id/fragment2"
        android:layout_weight="1"
        android:layout_width="0px"
        android:layout_height="match_parent" />
    -->
</LinearLayout>
```

2. Add the following code in bold to the `FragmentsActivity.java` file:

```
package net.learn2develop.Fragments;

import android.app.Activity;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.os.Bundle;
import android.view.Display;
import android.view.WindowManager;

public class FragmentsActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        FragmentManager fragmentManager = getFragmentManager();
        FragmentTransaction fragmentTransaction =
                fragmentManager.beginTransaction();

        //---get the current display info---
        WindowManager wm = getWindowManager();
        Display d = wm.getDefaultDisplay();
        if (d.getWidth() > d.getHeight())
        {
            //---landscape mode---
```

```
            Fragment1 fragment1 = new Fragment1();
            // android.R.id.content refers to the content
            // view of the activity
            fragmentTransaction.replace(
                    android.R.id.content, fragment1);
        }
        else
        {
            //---portrait mode---
            Fragment2 fragment2 = new Fragment2();
            fragmentTransaction.replace(
                    android.R.id.content, fragment2);
        }
        fragmentTransaction.commit();
    }
}
```

3. Press F11 to run the application on the Android emulator. Observe that when the emulator is in portrait mode, fragment 2 (yellow) is displayed (see Figure 2-20). If you press Ctrl+F11 to change the orientation of the emulator to landscape, fragment 1 (green) is shown instead (see Figure 2-21).

FIGURE 2-20

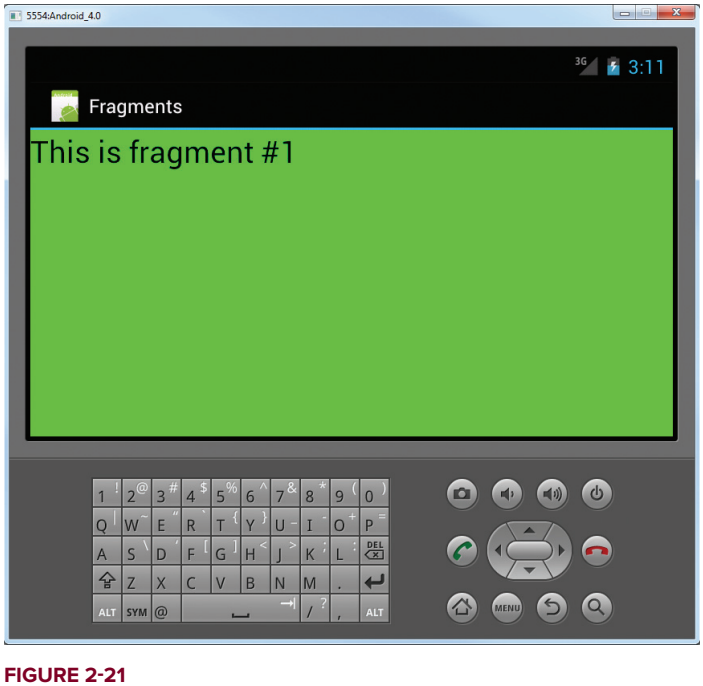


FIGURE 2-21

How It Works

To add fragments to an activity, you use the `FragmentManager` class by first obtaining an instance of it:

```
FragmentManager fragmentManager = getFragmentManager();
```

You also need to use the `FragmentTransaction` class to perform fragment transactions in your activity (such as add, remove or replace):

```
FragmentTransaction fragmentTransaction =
        fragmentManager.beginTransaction();
```

In this example, you used the `WindowManager` to determine whether the device is currently in portrait mode or landscape mode. Once that is determined, you add the appropriate fragment to the activity by creating the fragment and then calling the `replace()` method of the `FragmentTransaction` object to add the fragment to the specified view container (in this case, `android.R.id.content` refers to the content view of the activity):

```
//---landscape mode---
Fragment1 fragment1 = new Fragment1();
// android.R.id.content refers to the content
// view of the activity
fragmentTransaction.replace(
        android.R.id.content, fragment1);
```

Using the `replace()` method is essentially the same as calling the `remove()` method followed by the `add()` method of the `FragmentTransaction` object. To ensure that the changes take effect, you need to call the `commit()` method:

```
fragmentTransaction.commit();
```

Life Cycle of a Fragment

Like activities, fragments have their own life cycle. Understanding the life cycle of a fragment enables you to properly save an instance of the fragment when it is destroyed, and restore it to its previous state when it is recreated.

The following Try It Out examines the various states experienced by a fragment.

TRY IT OUT Understanding the Life Cycle of a Fragment

codefile Fragments.zip available for download at Wrox.com

1. Using the same project created in the previous section, add the following code in bold to the `Fragment1.java` file:

```
package net.learn2develop.Fragments;

import android.app.Activity;
import android.app.Fragment;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
```

```
import android.view.ViewGroup;

public class Fragment1 extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
    ViewGroup container, Bundle savedInstanceState) {

        Log.d("Fragment 1", "onCreateView");

        //---Inflate the layout for this fragment---
        return inflater.inflate(
            R.layout.fragment1, container, false);
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        Log.d("Fragment 1", "onAttach");
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d("Fragment 1", "onCreate");
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        Log.d("Fragment 1", "onActivityCreated");
    }

    @Override
    public void onStart() {
        super.onStart();
        Log.d("Fragment 1", "onStart");
    }

    @Override
    public void onResume() {
        super.onResume();
        Log.d("Fragment 1", "onResume");
    }

    @Override
    public void onPause() {
        super.onPause();
        Log.d("Fragment 1", "onPause");
    }

    @Override
    public void onStop() {
        super.onStop();
        Log.d("Fragment 1", "onStop");
```

```
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        Log.d("Fragment 1", "onDestroyView");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d("Fragment 1", "onDestroy");
    }

    @Override
    public void onDetach() {
        super.onDetach();
        Log.d("Fragment 1", "onDetach");
    }

}
```

2. Switch the Android emulator to landscape mode by pressing Ctrl+F11.
3. Press F11 in Eclipse to debug the application on the Android emulator.
4. When the application is loaded on the emulator, the following is displayed in the LogCat window (Windows ➪ Show View ➪ LogCat):

```
12-09 04:17:43.436: D/Fragment 1(2995): onAttach
12-09 04:17:43.466: D/Fragment 1(2995): onCreate
12-09 04:17:43.476: D/Fragment 1(2995): onCreateView
12-09 04:17:43.506: D/Fragment 1(2995): onActivityCreated
12-09 04:17:43.506: D/Fragment 1(2995): onStart
12-09 04:17:43.537: D/Fragment 1(2995): onResume
```

5. Click the Home button on the emulator. The following output will be displayed in the LogCat window:

```
12-09 04:18:47.696: D/Fragment 1(2995): onPause
12-09 04:18:50.346: D/Fragment 1(2995): onStop
```

6. On the emulator, click the Home button and hold it. Launch the application again. This time, the following is displayed:

```
12-09 04:20:08.726: D/Fragment 1(2995): onStart
12-09 04:20:08.766: D/Fragment 1(2995): onResume
```

7. Finally, click the Back button on the emulator. Now you should see the following output:

```
12-09 04:21:01.426: D/Fragment 1(2995): onPause
12-09 04:21:02.346: D/Fragment 1(2995): onStop
12-09 04:21:02.346: D/Fragment 1(2995): onDestroyView
```

```
12-09 04:21:02.346: D/Fragment 1(2995): onDestroy
12-09 04:21:02.346: D/Fragment 1(2995): onDetach
```

How It Works

Like activities, fragments in Android also have their own life cycle. As you have seen, when a fragment is being created, it goes through the following states:

- `onAttach()`
- `onCreate()`
- `onCreateView()`
- `onActivityCreated()`

When the fragment becomes visible, it goes through these states:

- `onStart()`
- `onResume()`

When the fragment goes into the background mode, it goes through these states:

- `onPause()`
- `onStop()`

When the fragment is destroyed (when the activity it is currently hosted in is destroyed), it goes through the following states:

- `onPause()`
- `onStop()`
- `onDestroyView()`
- `onDestroy()`
- `onDetach()`

Like activities, you can restore an instance of a fragment using a `Bundle` object, in the following states:

- `onCreate()`
- `onCreateView()`
- `onActivityCreated()`

NOTE *You can save a fragment's state in the* `onSaveInstanceState()` *method. Chapter 3 discusses this topic in more detail.*

Most of the states experienced by a fragment are similar to those of activities. However, a few new states are specific to fragments:

- `onAttached()` — Called when the fragment has been associated with the activity
- `onCreateView()` — Called to create the view for the fragment

- `onActivityCreated()` — Called when the activity's `onCreate()` method has been returned
- `onDestroyView()` — Called when the fragment's view is being removed
- `onDetach()` — Called when the fragment is detached from the activity

Note one of the main differences between activities and fragments: When an activity goes into the background, the activity is placed in the back stack. This allows the activity to be resumed when the user presses the Back button. In the case of fragments, however, they are not automatically placed in the back stack when they go into the background. Rather, to place a fragment into the back stack, you need to explicitly call the `addToBackStack()` method during a fragment transaction, like this:

```
//---get the current display info---
WindowManager wm = getWindowManager();
Display d = wm.getDefaultDisplay();
if (d.getWidth() > d.getHeight())
{
    //---landscape mode---
    Fragment1 fragment1 = new Fragment1();
    // android.R.id.content refers to the content
    // view of the activity
    fragmentTransaction.replace(
            android.R.id.content, fragment1);
}
else
{
    //---portrait mode---
    Fragment2 fragment2 = new Fragment2();
    fragmentTransaction.replace(
            android.R.id.content, fragment2);
}
//---add to the back stack---
fragmentTransaction.addToBackStack(null);
fragmentTransaction.commit();
```

The preceding code ensures that after the fragment has been added to the activity, the user can click the Back button to remove it.

Interactions between Fragments

Very often, an activity may contain one or more fragments working together to present a coherent UI to the user. In this case, it is very important for fragments to communicate with one another and exchange data. For example, one fragment might contain a list of items (such as postings from an RSS feed) and when the user taps on an item in that fragment, details about the selected item may be displayed in another fragment.

The following Try It Out shows how one fragment can access the views contained within another fragment.

TRY IT OUT Communication between Fragments

1. Using the same project created in the previous section, add the following statement in bold to the `Fragment1.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#00FF00" >
<TextView
    android:id="@+id/lblFragment1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="This is fragment #1"
    android:textColor="#000000"
    android:textSize="25sp" />
</LinearLayout>
```

2. Add the following lines in bold to `fragment2.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#FFFE00" >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="This is fragment #2"
    android:textColor="#000000"
    android:textSize="25sp" />

<Button
    android:id="@+id/btnGetText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Get text in Fragment #1"
    android:textColor="#000000"
    android:onClick="onClick" />

</LinearLayout>
```

3. Put back the two fragments in `main.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >

    <fragment
        android:name="net.learn2develop.Fragments.Fragment1"
        android:id="@+id/fragment1"
        android:layout_weight="1"
        android:layout_width="0px"
        android:layout_height="match_parent" />
    <fragment
        android:name="net.learn2develop.Fragments.Fragment2"
        android:id="@+id/fragment2"
        android:layout_weight="1"
        android:layout_width="0px"
        android:layout_height="match_parent" />

</LinearLayout>
```

4. Modify the `FragmentsActivity.java` file by commenting out the code that you added in the earlier sections. It should look like this after modification:

```
public class FragmentsActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        /*
        FragmentManager fragmentManager = getFragmentManager();
        FragmentTransaction fragmentTransaction =
                fragmentManager.beginTransaction();

        //---get the current display info---
        WindowManager wm = getWindowManager();
        Display d = wm.getDefaultDisplay();
        if (d.getWidth() > d.getHeight())
        {
            //---landscape mode---
            Fragment1 fragment1 = new Fragment1();
            // android.R.id.content refers to the content
            // view of the activity
            fragmentTransaction.replace(
                    android.R.id.content, fragment1);
        }
        else
        {
            //---portrait mode---
            Fragment2 fragment2 = new Fragment2();
            fragmentTransaction.replace(
                    android.R.id.content, fragment2);
        }
```

```
        //---add to the back stack---
        fragmentTransaction.addToBackStack(null);
        fragmentTransaction.commit();
        */
    }
}
```

5. Add the following statements in bold to the `Fragment2.java` file:

```
package net.learn2develop.Fragments;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class Fragment2 extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
    ViewGroup container, Bundle savedInstanceState) {
        //---Inflate the layout for this fragment---
        return inflater.inflate(
            R.layout.fragment2, container, false);
    }

    @Override
    public void onStart() {
        super.onStart();
        //---Button view---
        Button btnGetText = (Button)
            getActivity().findViewById(R.id.btnGetText);
        btnGetText.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                TextView lbl = (TextView)
                    getActivity().findViewById(R.id.lblFragment1);
                Toast.makeText(getActivity(), lbl.getText(),
                    Toast.LENGTH_SHORT).show();
            }
        });
    }

}
```

6. Press F11 to debug the application on the Android emulator. In the second fragment on the right, click the button. You should see the `Toast` class displaying the text "This is fragment #1" (see Figure 2-22).

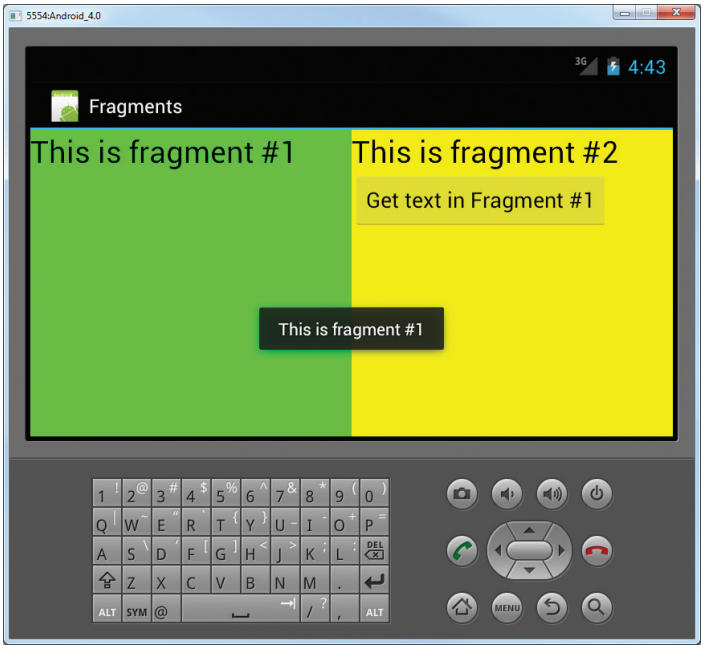


FIGURE 2-22

How It Works

Because fragments are embedded within activities, you can obtain the activity in which a fragment is currently embedded by first using the `getActivity()` method and then using the `findViewById()` method to locate the view(s) contained within the fragment:

```
TextView lbl = (TextView)
    getActivity().findViewById(R.id.lblFragment1);
Toast.makeText(getActivity(), lbl.getText(),
    Toast.LENGTH_SHORT).show();
```

The `getActivity()` method returns the activity with which the current fragment is currently associated.

Alternatively, you can also add the following method to the `FragmentsActivity.java` file:

```
public void onClick(View v) {
    TextView lbl = (TextView)
        findViewById(R.id.lblFragment1);
    Toast.makeText(this, lbl.getText(),
        Toast.LENGTH_SHORT).show();
}
```

CALLING BUILT-IN APPLICATIONS USING INTENTS

Until this point, you have seen how to call activities within your own application. One of the key aspects of Android programming is using the intent to call activities from other applications. In particular, your application can call the many built-in applications that are included with an Android device. For example, if your application needs to load a web page, you can use the `Intent` object to invoke the built-in web browser to display the web page, instead of building your own web browser for this purpose.

The following Try It Out demonstrates how to call some of the built-in applications commonly found on an Android device.

TRY IT OUT Calling Built-In Applications Using Intents

codefile Intents.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **Intents**.
2. Add the following statements in bold to the `main.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<Button
    android:id="@+id/btn_webbrowser"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Web Browser"
    android:onClick="onClickWebBrowser" />

<Button
    android:id="@+id/btn_makecalls"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Make Calls"
    android:onClick="onClickMakeCalls"  />

<Button
    android:id="@+id/btn_showMap"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Show Map"
    android:onClick="onClickShowMap" />

</LinearLayout>
```

3. Add the following statements in bold to the `IntentsActivity.java` file:

```
package net.learn2develop.Intents;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;

public class IntentsActivity extends Activity {

    int request_Code = 1;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClickWebBrowser(View view) {
        Intent i = new
                Intent(android.content.Intent.ACTION_VIEW,
                        Uri.parse("http://www.amazon.com"));
        startActivity(i);
    }

    public void onClickMakeCalls(View view) {
        Intent i = new
                Intent(android.content.Intent.ACTION_DIAL,
                        Uri.parse("tel:+651234567"));
        startActivity(i);
    }

    public void onClickShowMap(View view) {
        Intent i = new
                Intent(android.content.Intent.ACTION_VIEW,
                        Uri.parse("geo:37.827500,-122.481670"));
        startActivity(i);
    }
}
```

4. Press F11 to debug the application on the Android emulator.

5. Click the Web Browser button to load the Browser application on the emulator. Figure 2-23 shows the built-in Browser application displaying the site www.amazon.com.

6. Click the Make Calls button and the Phone application, as shown in Figure 2-24, will load.

FIGURE 2-23

FIGURE 2-24

7. Similarly, to load the Maps application, shown in Figure 2-25, click the Show Map button.

NOTE *In order to display the Maps application, you need to run the application on an AVD that supports the Google APIs.*

How It Works

In this example, you saw how you can use the `Intent` class to invoke some of the built-in applications in Android (such as Maps, Phone, Contacts, and Browser).

In Android, intents usually come in pairs: *action* and *data*. The *action* describes what is to be performed, such as editing an item, viewing the content of an item, and so on. The *data* specifies what is affected, such as a person in the Contacts database. The data is specified as an `Uri` object.

Some examples of action are as follows:

- `ACTION_VIEW`
- `ACTION_DIAL`
- `ACTION_PICK`

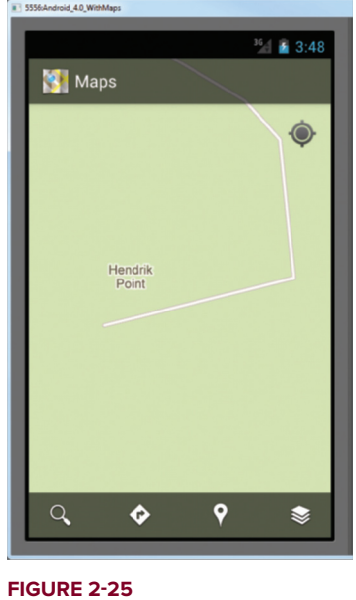


FIGURE 2-25

Some examples of data include the following:

- `www.google.com`
- `tel:+651234567`
- `geo:37.827500,-122.481670`
- `content://contacts`

NOTE *The section "Using Intent Filters" explains the type of data you can define for use in an activity.*

Collectively, the action and data pair describes the operation to be performed. For example, to dial a phone number, you would use the pair `ACTION_DIAL/tel:+651234567`. To display a list of contacts stored in your phone, you use the pair `ACTION_VIEW/content://contacts`. To pick a contact from the list of contacts, you use the pair `ACTION_PICK/content://contacts`.

In the first button, you create an `Intent` object and then pass two arguments to its constructor, the action and the data:

```
Intent i = new
        Intent(android.content.Intent.ACTION_VIEW,
                Uri.parse("http://www.amazon.com"));
startActivity(i);
```

The action here is represented by the `android.content.Intent.ACTION_VIEW` constant. You use the `parse()` method of the `Uri` class to convert a URL string into a `Uri` object.

The `android.content.Intent.ACTION_VIEW` constant actually refers to the "`android.intent.action.VIEW`" action, so the preceding could be rewritten as follows:

```
Intent i = new
        Intent("android.intent.action.VIEW",
                Uri.parse("http://www.amazon.com"));
startActivity(i);
```

The preceding code snippet can also be rewritten like this:

```
Intent i = new
        Intent("android.intent.action.VIEW");
i.setData(Uri.parse("http://www.amazon.com"));
startActivity(i);
```

Here, you set the data separately using the `setData()` method.

For the second button, you dial a specific number by passing in the telephone number in the data portion:

```
Intent i = new
        Intent(android.content.Intent.ACTION_DIAL,
                Uri.parse("tel:+651234567"));
startActivity(i);
```

In this case, the dialer will display the number to be called. The user must still press the dial button to dial the number. If you want to directly call the number without user intervention, change the action as follows:

```
Intent i = new
        Intent(android.content.Intent.ACTION_CALL,
                Uri.parse("tel:+651234567"));
startActivity(i);
```

NOTE *If you want your application to directly call the specified number, you need to add the* `android.permission.CALL_PHONE` *permission to your application.*

To display the dialer without specifying any number, simply omit the data portion, like this:

```
Intent i = new
        Intent(android.content.Intent.ACTION_DIAL);
startActivity(i);
```

The third button displays a map using the `ACTION_VIEW` constant:

```
Intent i = new
        Intent(android.content.Intent.ACTION_VIEW,
                Uri.parse("geo:37.827500,-122.481670"));
startActivity(i);
```

Here, instead of using "`http`" you use the "`geo`" scheme.

Understanding the Intent Object

So far, you have seen the use of the `Intent` object to call other activities. This is a good time to recap and gain a more detailed understanding of how the `Intent` object performs its magic.

First, you learned that you can call another activity by passing its action to the constructor of an `Intent` object:

```
startActivity(new Intent("net.learn2develop.SecondActivity"));
```

The action (in this example "`net.learn2develop.SecondActivity`") is also known as the *component name*. This is used to identify the target activity/application that you want to invoke. You can also rewrite the component name by specifying the class name of the activity if it resides in your project, like this:

```
startActivity(new Intent(this, SecondActivity.class));
```

You can also create an `Intent` object by passing in an action constant and data, such as the following:

```
Intent i = new
        Intent(android.content.Intent.ACTION_VIEW,
                Uri.parse("http://www.amazon.com"));
startActivity(i);
```

The action portion defines what you want to do, while the data portion contains the data for the target activity to act upon. You can also pass the data to the `Intent` object using the `setData()` method:

```
Intent i = new
        Intent("android.intent.action.VIEW");
i.setData(Uri.parse("http://www.amazon.com"));
```

In this example, you indicate that you want to view a web page with the specified URL. The Android OS will look for all activities that are able to satisfy your request. This process is known as *intent resolution*. The next section discusses in more detail how your activities can be the target of other activities.

For some intents, there is no need to specify the data. For example, to select a contact from the Contacts application, you specify the action and then indicate the MIME type using the `setType()` method:

```
Intent i = new
    Intent(android.content.Intent.ACTION_PICK);
i.setType(ContactsContract.Contacts.CONTENT_TYPE);
```

NOTE *Chapter 7 discusses how to use the Contacts application from within your application.*

The `setType()` method explicitly specifies the MIME data type to indicate the type of data to return. The MIME type for `ContactsContract.Contacts.CONTENT_TYPE` is `"vnd.android.cursor.dir/contact"`.

Besides specifying the action, the data, and the type, an `Intent` object can also specify a category. A category groups activities into logical units so that Android can use it for further filtering. The next section discusses categories in more detail.

To summarize, an `Intent` object can contain the following information:

- Action
- Data
- Type
- Category

Using Intent Filters

Earlier, you saw how an activity can invoke another activity using the `Intent` object. In order for other activities to invoke your activity, you need to specify the action and category within the `<intent-filter>` element in the `AndroidManifest.xml` file, like this:

```
<intent-filter >
    <action android:name="net.learn2develop.SecondActivity" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

This is a very simple example in which one activity calls another using the "`net.learn2develop.SecondActivity`" action. The following Try It Out shows you a more sophisticated example.

TRY IT OUT Specifying Intent Filters in More Detail

1. Using the Intents project created earlier, add a new class to the project and name it `MyBrowserActivity`. Also add a new XML file to the `res/layout` folder and name it **browser.xml**.
2. Add the following statements in bold to the `AndroidManifest.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.Intents"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".IntentsActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".MyBrowserActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <action android:name="net.learn2develop.MyBrowser" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:scheme="http" />
            </intent-filter>
        </activity>

    </application>

</manifest>
```

3. Add the following statements in bold to the `main.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<Button
    android:id="@+id/btn_webbrowser"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Web Browser"
    android:onClick="onClickWebBrowser" />

<Button
    android:id="@+id/btn_makecalls"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Make Calls"
    android:onClick="onClickMakeCalls"  />

<Button
    android:id="@+id/btn_showMap"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Show Map"
    android:onClick="onClickShowMap" />

<Button
    android:id="@+id/btn_launchMyBrowser"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Launch My Browser"
    android:onClick="onClickLaunchMyBrowser" />

</LinearLayout>
```

4. Add the following statements in bold to the `IntentsActivity.java` file:

```
package net.learn2develop.Intents;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;

public class IntentsActivity extends Activity {

    int request_Code = 1;

    /** Called when the activity is first created. */
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) { … }

    public void onClickWebBrowser(View view) { … }

    public void onClickMakeCalls(View view) { ... }

    public void onClickShowMap(View view) { ... }

    public void onClickLaunchMyBrowser(View view) {
        Intent i = new
                Intent("net.learn2develop.MyBrowser");
        i.setData(Uri.parse("http://www.amazon.com"));
        startActivity(i);
    }

}
```

5. Add the following statements in bold to the `browser.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
<WebView
    android:id="@+id/WebView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</LinearLayout>
```

6. Add the following statements in bold to the `MyBrowserActivity.java` file:

```
package net.learn2develop.Intents;

import android.app.Activity;
import android.net.Uri;
import android.os.Bundle;
import android.webkit.WebView;
import android.webkit.WebViewClient;

public class MyBrowserActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.browser);

        Uri url = getIntent().getData();
        WebView webView = (WebView) findViewById(R.id.WebView01);
        webView.setWebViewClient(new Callback());
```

```
            webView.loadUrl(url.toString());
        }

        private class Callback extends WebViewClient {
            @Override
            public boolean shouldOverrideUrlLoading
            (WebView view, String url) {
                return(false);
            }
        }
    }
```

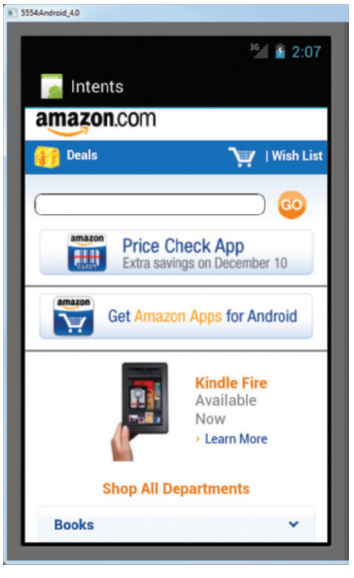


FIGURE 2-26

7. Press F11 to debug the application on the Android emulator.
8. Click the Launch my Browser button and you should see the new activity displaying the Amazon.com web page (see Figure 2-26).

How It Works

In this example, you created a new activity named `MyBrowserActivity`. You first needed to declare it in the `AndroidManifest.xml` file:

```
<activity android:name=".MyBrowserActivity"
          android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <action android:name="net.learn2develop.MyBrowser" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http" />
    </intent-filter>
</activity>
```

In the `<intent-filter>` element, you declared it to have two actions, one category, and one data. This means that all other activities can invoke this activity using either the "`android.intent.action.VIEW`" or the "`net.learn2develop.MyBrowser`" action. For all activities that you want others to call using the `startActivity()` or `startActivityForResult()` methods, they need to have the "`android.intent.category.DEFAULT`" category. If not, your activity will not be callable by others. The `<data>` element specifies the type of data expected by the activity. In this case, it expects the data to start with the "`http://`" prefix.

The preceding intent filter could also be rewritten as follows:

```
<activity android:name=".MyBrowserActivity"
          android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http" />
    </intent-filter>
    <intent-filter>
        <action android:name="net.learn2develop.MyBrowser" />
        <category android:name="android.intent.category.DEFAULT" />
```

```
            <data android:scheme="http" />
        </intent-filter>
    </activity>
```

Writing the intent filter this way makes it much more readable, and it logically groups the action, category, and data within an intent filter.

If you now use the `ACTION_VIEW` action with the data shown here, Android will display a selection (as shown in Figure 2-27):

```
Intent i = new
        Intent(android.content.Intent.ACTION_VIEW,
            Uri.parse("http://www.amazon.com"));
```

You can choose between using the Browser application or the Intents application that you are currently building.

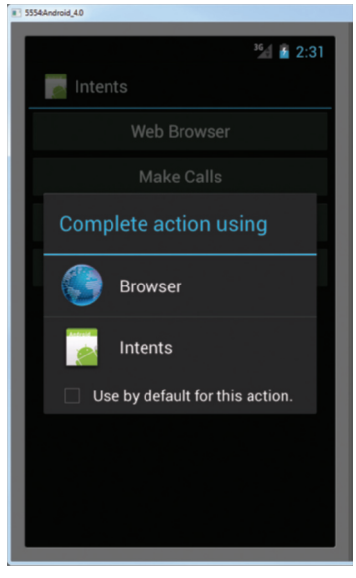


FIGURE 2-27

Notice that when multiple activities match your `Intent` object, the dialog titled "Complete action using" appears. You can customize this by using the `createChooser()` method from the `Intent` class, like this:

```
Intent i = new
        Intent(android.content.Intent.ACTION_VIEW,
            Uri.parse("http://www.amazon.com"));
startActivity(Intent.createChooser(i, "Open URL using..."));
```

The preceding will change the dialog title to "Open URL using...," as shown in Figure 2-28. Note that the "Use by default for this action" option is now not available.

The added benefit of using the `createChooser()` method is that in the event that no activity matches your `Intent` object, your application will not crash. Instead, it will display the message shown in Figure 2-29.

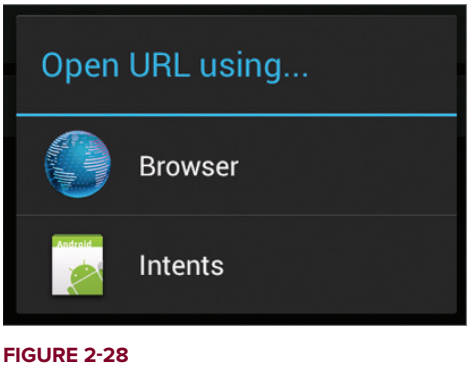


FIGURE 2-28

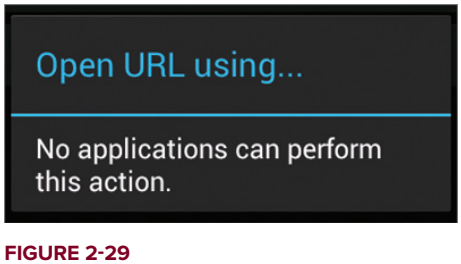


FIGURE 2-29

Adding Categories

You can group your activities into categories by using the `<category>` element in the intent filter. Suppose you have added the following `<category>` element to the `AndroidManifest.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.Intents"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".IntentsActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".MyBrowserActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <action android:name="net.learn2develop.MyBrowser" />
                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="net.learn2develop.Apps" />
                <data android:scheme="http" />
            </intent-filter>
        </activity>

    </application>

</manifest>
```

In this case, the following code will directly invoke the `MyBrowerActivity` activity:

```
Intent i = new
        Intent(android.content.Intent.ACTION_VIEW,
            Uri.parse("http://www.amazon.com"));
i.addCategory("net.learn2develop.Apps");
startActivity(Intent.createChooser(i, "Open URL using..."));
```

You add the category to the `Intent` object using the `addCategory()` method. If you omit the `addCategory()` statement, the preceding code will still invoke the `MyBrowerActivity` activity because it will still match the default category `android.intent.category.DEFAULT`.

However, if you specify a category that does not match the category defined in the intent filter, it will not work (no activity will be launched):

```
Intent i = new
        Intent(android.content.Intent.ACTION_VIEW,
            Uri.parse("http://www.amazon.com"));
//i.addCategory("net.learn2develop.Apps");
//---this category does not match any in the intent-filter---
i.addCategory("net.learn2develop.OtherApps");
startActivity(Intent.createChooser(i, "Open URL using..."));
```

The preceding category (`net.learn2develop.OtherApps`) does not match any category in the intent filter, so a run-time exception will be raised (if you don't use the `createChoose()` method of the `Intent` class).

If you add the following category in the intent filter of `MyBrowerActivity`, then the preceding code will work:

```
<activity android:name=".MyBrowserActivity"
          android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <action android:name="net.learn2develop.MyBrowser" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="net.learn2develop.Apps" />
        <category android:name="net.learn2develop.OtherApps" />
        <data android:scheme="http" />
    </intent-filter>
</activity>
```

You can add multiple categories to an `Intent` object; for example, the following statements add the `net.learn2develop.SomeOtherApps` category to the `Intent` object:

```
Intent i = new
        Intent(android.content.Intent.ACTION_VIEW,
            Uri.parse("http://www.amazon.com"));
//i.addCategory("net.learn2develop.Apps");
//---this category does not match any in the intent-filter---
i.addCategory("net.learn2develop.OtherApps");
i.addCategory("net.learn2develop.SomeOtherApps");
startActivity(Intent.createChooser(i, "Open URL using..."));
```

Because the intent filter does not define the `net.learn2develop.SomeOtherApps` category, the preceding code will not be able to invoke the `MyBrowerActivity` activity. To fix this, you need to add the `net.learn2develop.SomeOtherApps` category to the intent filter again.

From this example, it is evident that when using an `Intent` object with categories, all categories added to the `Intent` object must fully match those defined in the intent filter before an activity can be invoked.

DISPLAYING NOTIFICATIONS

So far, you have been using the `Toast` class to display messages to the user. While the `Toast` class is a handy way to show users alerts, it is not persistent. It flashes on the screen for a few seconds and then disappears. If it contains important information, users may easily miss it if they are not looking at the screen.

For messages that are important, you should use a more persistent method. In this case, you should use the `NotificationManager` to display a persistent message at the top of the device, commonly known as the *status bar* (sometimes also referred to as the *notification bar*). The following Try It Out demonstrates how.

TRY IT OUT Displaying Notifications on the Status Bar

codefile Notifications.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it **Notifications**.
2. Add a new class file named `NotificationView` to the package. In addition, add a new `notification.xml` file to the `res/layout` folder.
3. Populate the `notification.xml` file as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Here are the details for the notification..." />
</LinearLayout>
```

4. Populate the `NotificationView.java` file as follows:

```
package net.learn2develop.Notifications;

import android.app.Activity;
import android.app.NotificationManager;
import android.os.Bundle;

public class NotificationView extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.notification);

        //---look up the notification manager service---
        NotificationManager nm = (NotificationManager)
```

```
            getSystemService(NOTIFICATION_SERVICE);

        //---cancel the notification that we started---
        nm.cancel(getIntent().getExtras().getInt("notificationID"));
    }
}
```

5. Add the following statements in bold to the `AndroidManifest.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.Notifications"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />
    <uses-permission android:name="android.permission.VIBRATE"/>

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".NotificationsActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".NotificationView"
            android:label="Details of notification">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

6. Add the following statements in bold to the `main.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

<Button
    android:id="@+id/btn_displaynotif"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Display Notification"
    android:onClick="onClick"/>

</LinearLayout>
```

7. Finally, add the following statements in bold to the `NotificationsActivity.java` file:

```
package net.learn2develop.Notifications;

import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class NotificationsActivity extends Activity {
    int notificationID = 1;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClick(View view) {
        displayNotification();
    }

    protected void displayNotification()
    {
        //---PendingIntent to launch activity if the user selects
        // this notification---
        Intent i = new Intent(this, NotificationView.class);
        i.putExtra("notificationID", notificationID);

        PendingIntent pendingIntent =
            PendingIntent.getActivity(this, 0, i, 0);

        NotificationManager nm = (NotificationManager)
            getSystemService(NOTIFICATION_SERVICE);

        Notification notif = new Notification(
            R.drawable.ic_launcher,
            "Reminder: Meeting starts in 5 minutes",
            System.currentTimeMillis());

        CharSequence from = "System Alarm";
        CharSequence message = "Meeting with customer at 3pm...";

        notif.setLatestEventInfo(this, from, message, pendingIntent);

        //---100ms delay, vibrate for 250ms, pause for 100 ms and
        // then vibrate for 500ms---
        notif.vibrate = new long[] { 100, 250, 100, 500};
        nm.notify(notificationID, notif);
    }

}
```

8. Press F11 to debug the application on the Android emulator.
9. Click the Display Notification button and a notification ticker text (set in the constructor of the `Notification` object) will appear on the status bar (see Figure 2-30).
10. Clicking and dragging the status bar down will reveal the notification details set using the `setLatestEventInfo()` method of the `Notification` object (see Figure 2-31).
11. Clicking on the notification will reveal the `NotificationView` activity (see Figure 2-32). This also causes the notification to be dismissed from the status bar.

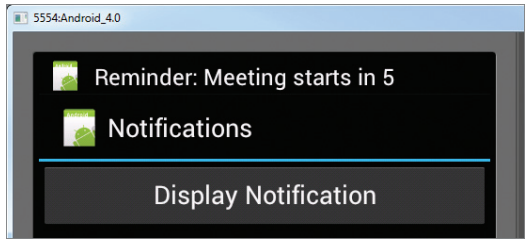


FIGURE 2-30

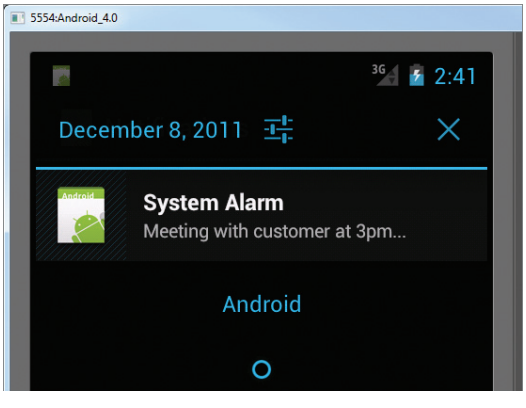


FIGURE 2-31

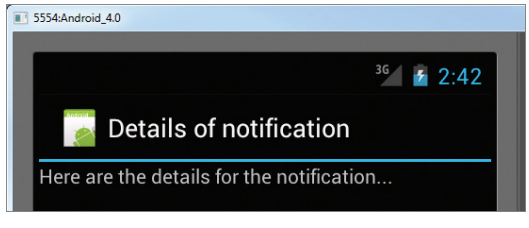


FIGURE 2-32

How It Works

To display a notification, you first created an `Intent` object to point to the `NotificationView` class:

```
Intent i = new Intent(this, NotificationView.class);
i.putExtra("notificationID", notificationID);
```

This intent is used to launch another activity when the user selects a notification from the list of notifications. In this example, you added a name/value pair to the `Intent` object so that you can tag the notification ID, identifying the notification to the target activity. This ID will be used to dismiss the notification later.

You also need to create a `PendingIntent` object. A `PendingIntent` object helps you to perform an action on your application's behalf, often at a later time, regardless of whether your application is running or not. In this case, you initialized it as follows:

```
PendingIntent pendingIntent =
    PendingIntent.getActivity(this, 0, i, 0);
```

The `getActivity()` method retrieves a `PendingIntent` object and you set it using the following arguments:

- **context** — Application context
- **request code** — Request code for the intent
- **intent** — The intent for launching the target activity
- **flags** — The flags in which the activity is to be launched

You then obtain an instance of the `NotificationManager` class and create an instance of the `Notification` class:

```
NotificationManager nm = (NotificationManager)
    getSystemService(NOTIFICATION_SERVICE);

Notification notif = new Notification(
    R.drawable.ic_launcher,
    "Reminder: Meeting starts in 5 minutes",
    System.currentTimeMillis());
```

The `Notification` class enables you to specify the notification's main information when the notification first appears on the status bar. The second argument to the `Notification` constructor sets the "ticker text" on the status bar (see Figure 2-33).

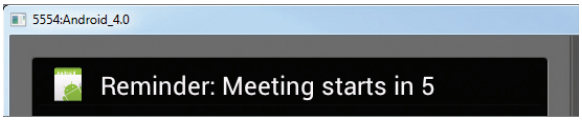


FIGURE 2-33

Next, you set the details of the notification using the `setLatestEventInfo()` method:

```
CharSequence from = "System Alarm";
CharSequence message = "Meeting with customer at 3pm...";

notif.setLatestEventInfo(this, from, message, pendingIntent);

//---100ms delay, vibrate for 250ms, pause for 100 ms and
// then vibrate for 500ms---
notif.vibrate = new long[] { 100, 250, 100, 500};
```

The preceding also sets the notification to vibrate the phone. Finally, to display the notification you use the `notify()` method:

```
nm.notify(notificationID, notif);
```

When the user clicks on the notification, the `NotificationView` activity is launched. Here, you dismiss the notification by using the `cancel()` method of the `NotificationManager` object and passing it the ID of the notification (passed in via the `Intent` object):

```
//---look up the notification manager service---
NotificationManager nm = (NotificationManager)
    getSystemService(NOTIFICATION_SERVICE);

//---cancel the notification that we started---
nm.cancel(getIntent().getExtras().getInt("notificationID"));
```

SUMMARY

This chapter first provided a detailed look at how activities and fragments work and the various forms in which you can display them. You also learned how to display dialog windows using activities.

The second part of this chapter demonstrated a very important concept in Android — the intent. The intent is the "glue" that enables different activities to be connected, and it is a vital concept to understand when developing for the Android platform.

EXERCISES

1. What will happen if you have two or more activities with the same intent filter action name?
2. Write the code to invoke the built-in Browser application.
3. Which components can you specify in an intent filter?
4. What is the difference between the `Toast` class and the `NotificationManager` class?
5. Name the two ways to add fragments to an activity.
6. Name one key difference between a fragment and an activity.

Answers to the exercises can be found in Appendix C.

▶ WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
Creating an activity	All activities must be declared in the `AndroidManifest.xml` file.
Key life cycle of an activity	When an activity is started, the `onStart()` and `onResume()` events are always called.
	When an activity is killed or sent to the background, the `onPause()` event is always called.
Displaying an activity as a dialog	Use the `showDialog()` method and implement the `onCreateDialog()` method.
Fragments	Fragments are "mini-activities" that can be added or removed from activities.
Manipulating fragments programmatically	You need to use the `FragmentManager` and `FragmentTransaction` classes when adding, removing, or replacing fragments during runtime.
Life cycle of a fragment	Similar to that of an activity — you save the state of a fragment in the `onPause()` event, and restore its state in one of the following events: `onCreate()`, `onCreateView()`, or `onActivityCreated()`.
Intent	The "glue" that connects different activities
Intent filter	The "filter" that enables you to specify how your activities should be called
Calling an activity	Use the `startActivity()` or `startActivityForResult()` method.
Passing data to an activity	Use the `Bundle` object.
Components in an `Intent` object	An `Intent` object can contain the following: action, data, type, and category.
Displaying notifications	Use the `NotificationManager` class.
`PendingIntent` object	A `PendingIntent` object helps you to perform an action on your application's behalf, often at a later time, regardless of whether or not your application is running.

3

Getting to Know the Android User Interface

WHAT YOU WILL LEARN IN THIS CHAPTER

- The various ViewGroups you can use to lay out your views
- How to adapt and manage changes in screen orientation
- How to create the UI programmatically
- How to listen for UI notifications

In Chapter 2, you learned about the `Activity` class and its life cycle. You learned that an activity is a means by which users interact with the application. However, an activity by itself does not have a presence on the screen. Instead, it has to draw the screen using *Views* and *ViewGroups*. In this chapter, you will learn the details about creating user interfaces in Android, and how users interact with them. In addition, you will learn how to handle changes in screen orientation on your Android devices.

UNDERSTANDING THE COMPONENTS OF A SCREEN

In Chapter 2, you learned that the basic unit of an Android application is an *activity*. An *activity* displays the user interface of your application, which may contain widgets such as buttons, labels, textboxes, and so on. Typically, you define your UI using an XML file (e.g., the `main.xml` file located in the `res/layout` folder of your project), which looks similar to the following:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```