Making Everything Easier!™

Learn to:

- **Create apps for hot smartphones like Droid™ X, Galaxy S, and MyTouch®**
- **Download the SDK and get Eclipse up and running**
- **Code Android applications**
- **Submit your apps to the Android Market**

Donn Felker

Independent software development consultant

Get More and Do More at

Dummies.com® Start with **FREE** Cheat Sheets

Cheat Sheets include
- Checklists
- Charts
- Common Instructions
- And Other Good Stuff!

To access the Cheat Sheet created specifically for this book, go to *www.dummies.com/cheatsheet/androidapplicationdevelopment*

Get Smart at Dummies.com

Dummies.com makes your life easier with 1,000s of answers on everything from removing wallpaper to using the latest version of Windows.

Check out our
- Videos
- Illustrated Articles
- Step-by-Step Instructions

Plus, each month you can win valuable prizes by entering our Dummies.com sweepstakes. *

Want a weekly dose of Dummies? Sign up for Newsletters on
- Digital Photography
- Microsoft Windows & Office
- Personal Finance & Investing
- Health & Wellness
- Computing, iPods & Cell Phones
- eBay
- Internet
- Food, Home & Garden

Find out “HOW” at Dummies.com

**Sweepstakes not currently available in all countries; visit Dummies.com for official rules.*

by Donn

Felker with Joshua Dobbs

Android™ Application Development For Dummies®
Published by
Wiley Publishing, Inc.

111 River Street
Hoboken, NJ 07030-5774
www.wiley.com

Published by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

Library of Congress Control Number: 2010939962

ISBN: 978-0-470-77018-4

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

About the Authors

Donn Felker is a recognized leader in the development and consultation of state-of-the-art, cutting-edge software in the mobile and Web fi elds. He is an independent consultant with over 10 years of professional experience in various markets that include entertainment, health, retail, insurance, fi nan cial, and real estate. He is a mobile junkie, serial entrepreneur, and creative innovator in all things mobile and Web. He is the founder of Agilevent, an innovative creative development fi rm that has done work for small startups as well as Fortune 500 companies. He is a Microsoft ASP Insider, an MCTS for .NET Framework 2.0

and 3.5 Web Applications, and a certifi ed ScrumMaster. He's a national speaker on topics that include Android, .NET, and software architecture. He is the author of the TekPub.com Introduction to Android video series. He is a writer, presenter, and consultant on various topics rang ing from architecture to development in general, agile practices, and patterns and practices. Follow Donn on Twitter (@donnfelker) or read his blog here: http://blog.donnfelker.com.

Joshua Dobbs is a senior lead Web application developer for a large elec tronics manufacturer in Southern California. He has more than ten years' experience in Web and desktop application development. Josh was an early adopter of the Android platform and creates Android apps in his spare time. His apps have been downloaded more than 6 million times, and he was selected by Google as top Android developer for its Device Seeding Program. His Web site is www.joshdobbs.com.

Dedication

To my dogs, Lulu and Macho, and my cat, Vito: Thanks for keeping me company in the cold basement while I cranked out page after page in the wee hours of the morning while everyone else was asleep. Writing is a lonely gig, and your company helped the time pass much easier (and kept my feet and lap warm too).

To my dearest daughter, Sophia, who made even the toughest days brighter through her contagious, infectious laughter and antics. I love you.

Most of all, to my gorgeous wife, Ginamarie, who has always been very sup portive of all my crazy, harebrained ideas over the years. I would not have gotten where I am in my life if it were not for your support. I love you.

Author's Acknowledgments

Thanks to coauthor Joshua Dobbs for writing the couple of chapters that I needed help with. May we both have many more successful books in the future!

Thanks to Wiley Acquisitions Editor Kyle Looper for giving me a shot at writ ing this book. I really appreciate the help, support, and insight into everything publishing-related. You've been a life saver on this project. Thank you.

Project Editor Kathy Simpson pushed me beyond what I thought would be possible in terms of the organization of the content and readability. Thank you for being a diligent editor.

Copy Editor John Edwards helped fi nd some of my most subtle mistakes, which allowed me to polish the book content even more. Thank you.

Technical Editor Andre Taddeini is one of the few technical individuals I trust wholeheartedly. I'm glad you were my second pair of eyes on this project. Your sanity check of the technical accuracy of the book was outstanding. Thank you.

Finally, thank you to my friend John Benda for contributing by being supportive of me and my family during this process. It's been a long road. Now it's your turn to write a book!

Publisher's Acknowledgments

We're proud of this book; please send us your comments at http://dummies.custhelp.com. For other comments, please contact our Customer Care Department within the U.S. at 877-762-2974, outside the U.S. at 317-572-3993, or fax 317-572-4002.

Some of the people who helped bring this book to market include the following:

Acquisitions and Editorial

Project Editor: Kathy Simpson

Acquisitions Editor: Kyle Looper

Copy Editor: John Edwards

Technical Editor: Andre Taddeini

Editorial Manager: Jodi Jensen

Editorial Assistant: Amanda Graham

Sr. Editorial Assistant: Cherie Case

Cartoons: Rich Tennant (www.the5thwave.com)

Publishing and Editorial for Technology Dummies

Composition Services

Project Coordinator: Sheree Montgomery

Layout and Graphics: Nikki Gately, Laura Westhuis

Proofreaders: Laura Bowman, Rebecca Denoncour

Indexer: BIM Indexing & Proofreading Services

Richard Swadley, Vice President and Executive Group Publisher

Andy Cummings, Vice President and Publisher

Mary Bednarek, Executive Acquisitions Director

Mary C. Corder, Editorial Director

Publishing for Consumer Dummies

Diane Graves Steele, Vice President and Publisher

Composition Services

Gerry Fahey, Vice President of Production Services

Debbie Stailey, Director of Composition Services

Contents at a Glance

Table of Contents

Chapter 2: Prepping Your Development Headquarters 25

Introduction

Welcome to *Android Application Development For Dummies,* the first *For Dummies* book that covers Android application development. When I was contacted to write this book, I was ecstatic about the opportunity to spread the wealth of knowledge that I'd picked up over the past year and a half of Android development. I hope you enjoy finding out about how to pro gram for the Android platform from this book as much as I enjoyed writing it!

When Android was acquired by Google in 2005 (yes, Android was a start-up company at one point), I'll be honest, I didn't have much interest in it. I heard

that Google might be entering the mobile space, but as with anything in the technology industry, I didn't believe it until I saw it firsthand. Fast-forward to a few years later, when Google announced its first Android phone: the G1. When I heard this news, I was glued to the computer, reading reviews, watch ing videos, and researching the product as much as I could. I knew that this product would be the start of something huge.

I got my start in Android development about a week after my wife received her first G1 Android device. The G1 was the first publicly released Android device. It didn't match the rich feature set of the iPhone at the time, but I desperately believed in the platform. As soon as Donut (Android 1.6) was released, it was evident that Google was putting some effort into the product. Immediately after version 1.6 was released, talk of 2.0 was already on the horizon.

Today, we're on version 2.2 of the Android platform, and 3.0 is just around the corner. The platform is barely two years old, and I see no sign of the platform development slowing down. Without doubt, this is an exciting time in Android development. I hope that your excitement carries through as you read this book and later as you release your own applications on the market.

About This Book

Android Application Development For Dummies is a beginner's guide to devel oping Android applications. You don't need any Android application develop ment experience under your belt to get started. I expect you to approach this material as a blank slate because the Android platform accomplishes various mechanisms by using different paradigms that most programmers aren't used to using — or developing with — on a day-to-day basis. I expect you to be familiar with the Java programming language, however. You don't have to

be a Java guru, but you should understand the syntax, basic data structures, and language constructs. XML is also used in developing Android applica tions, so I advise understanding XML as well.

The Android platform is a *device-independent* platform, which means that you can develop applications for various devices. These devices include but aren't limited to phones, e-book readers, netbooks, and GPS devices. Soon, television sets will join the list. Yes, you read it correctly — TV! Google has announced plans to include a Google TV offering in the Android platform.

Finding out how to develop for the Android platform opens a large variety of development options for you. This book distills hundreds, if not thou sands, of pages of Android documentation, tips, tricks, and tutorials into a short, digestible format that allows you to springboard into your future as an Android developer. This book isn't a recipe book, but it gives you the basic knowledge to assemble various pieces of the Android framework to create interactive and compelling applications.

Conventions Used in This Book

Throughout the book, you use the Android framework classes, and you will be creating Java classes and XML files.

Code examples in this book appear in a monospace font so that they stand out from other text in the book. This means that the code you'll see looks like this:

```
public class MainActivity
```

Java is a high-level programming language that is case-sensitive, so be sure to enter the text into the editor *exactly* as you see it in the book. I also use the standard Java conventions in this book. Therefore, you can transition easily between my examples and the example code provided by the Android Software Development Kit (SDK). All class names, for example, appear in PascalCase format, and all class-scoped variables start with m.

All the URLs in the book appear in monospace font as well:

```
http://d.android.com
```

If you're ever unsure about anything in the code, you can download the full source code from my GitHub account, located at http://github.com/donnfelker. From time to time, I provide code updates to the source. You can also find other examples in my other source repositories stored on the same site. Finally, you can find the same material on the *For Dummies* Web site at www.dummies.com/go/androidappdevfd.

Foolish Assumptions

To begin programming with Android, you need a computer that runs one of the following operating systems:

✓ Windows XP (32 bit), Vista (32 or 64 bit), or Windows 7 (32 or 64 bit) ✓ Mac OS X (Intel) 10.5.8 (x86 only)

✓ Linux (i386)

You also need to download the Android SDK (which is free) and the Java Development Kit (or JDK, which is also free), if you don't already have them on your computer. I explain the entire installation process for all the tools and frameworks in Chapter 2.

As I state earlier in this introduction, because Android applications are developed in the Java programming language, you need to understand the Java language. Android also uses XML quite heavily to define various resources inside the application, so you should understand XML too. I don't expect you to be an expert in these languages, however. I started in Android with a background in C#, having done Java only in college nearly 10 years earlier, and I fared just fine.

You don't need a physical Android device, because all the applications you build in this book work on the emulator. I highly recommend developing on a

real device, however, because it allows you to interact with your applications as real users would.

How This Book Is Organized

Android Application Development For Dummies has four parts, which I describe in the following sections.

Part I: The Nuts and Bolts of Android

Part I introduces the tools and frameworks that you use to develop Android applications. It also introduces the various SDK components and shows you how they're used in the Android ecosystem.

Part II: Building and Publishing Your First Android Application

Part II introduces you to building your first Android application: the Silent Mode Toggle application. After you build the initial application, I show you how to create an app widget for the application that you can place on the home screen of the Android device. I tie everything together by demonstrat ing how to publish your application to the Android Market.

Part III: Creating a Feature-Rich Application

Part III takes your development skills up a notch by walking you through the construction of the Task Reminder application, which allows users to create various tasks with reminders. I cover the implementation of an SQLite data base in this multiscreen application. You also see how to use the Android status bar to create notifications that can help increase the usability of your application.

Part IV: The Part of Tens

Part IV brings together the prizes that I've found through my trials and tribu lations in Android development. I give you a tour of sample applications that prove to be stellar launching pads for your Android apps, and I introduce useful Android libraries that can make your Android development career a lot easier.

Icons Used in This Book

This icon indicates a useful pointer that you shouldn't skip.

This icon represents a friendly reminder about a vital point you should keep in mind while proceeding through a particular section of the chapter.

This icon signifies that the accompanying explanation may be informative but isn't essential to understanding Android application development. Feel free to skip these snippets, if you like.

This icon alerts you to potential problems that you may encounter along the way. Read and remember these tidbits to avoid possible trouble.

Where to Go from Here

It's time to explore the Android platform! If you're a bit nervous, let me assure you that you don't have to worry; you should be nervous only because you're excited.

Part I

The Nuts and Bolts of Android

In this part . . .

Part I introduces you to the Android platform and describes what makes a spectacular Android application. I briefly explore various parts of the Android software development kit (SDK) and explain how you can use them in your applications. I also guide you through the process of installing the tools and frameworks necessary to develop Android applications.

Chapter 1

Developing Spectacular Android Applications

In This Chapter

- Seeing reasons to develop Android apps
- Starting with the basics of Android programming
- Working with the hardware
- Getting familiar with the software

Google rocks! Google acquired the Android project in 2005 (see the side bar "The roots of Android" later in this chapter) to ensure that a mobile operating system (OS) could be created and maintained in an open platform. Google continues to pump time and resources into the Android project, which has already proved to be beneficial. As of July 2010, 160,000 Android handsets have been activated daily, which is good considering that handsets have been available only since October 2008. That's less than two years, and Android has already made a huge impact!

It has never been easier for a developer to be able to make money on his own. Android users may not know who you are, but they know what Google is, and they trust Google. Because your app resides in the Android Market — which Google controls — Google assumes that your application is okay too.

Why Develop for Android?

Well, the real question should be "Why not?" Do you want your app to be available to millions of users worldwide? Do you want to publish apps as soon as you're done writing and testing them? Do you like developing on open platforms? If you answered yes to any of these questions, I think you have your answer, but in case you're still undecided, keep reading, and I'll explain what I mean.

Market share

As a developer, you have an opportunity to develop apps for a fairly new market that is booming on a daily basis. Android is currently set to outpace many other carriers in market share in the industry in coming months. With so many users, it's never been easier to write an application that can be

downloaded and used by real people! The Android Market puts your app right into your users' hands easily! Users don't have to go searching the Internet to find an app to install. They just simply go to the Android Market that is preinstalled on their device, and they have access to all *your* apps. Because the Android Market comes preinstalled on most Android devices (I discuss a few exceptions later), users typically search the Android Market for all of their app needs. It's not hard to see an app's number of downloads soar in just a few days.

Time to market

With all the application programming interfaces (APIs) that Android comes packed with, it's easy to develop full-featured applications in a relatively short time frame. After you've signed up with the Android Market, just upload your apps and publish them. "Wait," you may say, "are you sure?" Why, yes, I am! Unlike other mobile marketplaces, the Android Market has no app-approval process. All you have to do is write apps and publish them.

Technically, anyone can publish anything, but it's good karma to keep within Google's terms of service and keep your apps family-friendly. Remember that Android users come from diverse areas of the world and are in all age categories.

Open platform

The Android operating system is *open platform,* meaning that it's not tied to one hardware manufacturer and/or one provider. As you can imagine, the openness of Android is allowing it to gain market share quickly. All hard ware manufacturers and providers can make and sell Android devices. The Android source code is available at http://source.android.com for you to view and/or modify. Nothing is holding you back from digging into the source code to see how a certain task is handled. The open-source code allows phone manufacturers to create custom user interfaces (UIs) and add built-in features to some devices. This also puts all developers on an even playing field. Everyone can access the raw Android source code.

11

The roots of Android

Most people don't know this, but Google didn't start the Android project. The initial Android operating system was created by a small start up company in Silicon Valley known as Android, Inc., which was purchased by Google in July 2005. The founders of Android, Inc., came from various Internet technology companies such as Danger, Wildfire Communications, T-Mobile, and WebTV. Google brought them into the Google team to help create what is now the full-fledged Android mobile operating system.

Cross-compatibility

Android can run on many devices with different screen sizes and resolutions. Besides being cross-compatible, Android comes with the tools that help you develop cross-compatible applications. Google allows your apps to run only on compatible devices. If your app requires a front-facing camera, for exam ple, only phones with a front-facing camera will be able to see your app in the Android Market. This arrangement is known as *feature detection.* (For more information on publishing your apps to the Android Market, see Chapter 8.)

For Android devices to be certified compatible (devices have to be compat ible to have access to the Android Market), they must follow certain hardware guidelines. These guidelines include but are not limited to the following:

- ✓ Camera
- ✓ Compass
- ✓ GPS (Global Positioning System) feature
- ✓ Bluetooth transceiver

See the Compatibility Program Overview page at http://source.android.com/compatibility/overview.html for specific device con figurations that are considered to be compatible. Compatibility ensures that your apps can run on all devices.

Mashup capability

A *mashup* combines two or more services to create an application. You can create a mashup by using the camera and Android's location services, for example, to take a picture with the exact location displayed on the image! It's easy to make a ton of apps by combining services or libraries in new and exciting ways.

With all the APIs that Android includes, it's easy to use two or more of these features to make your own app. You can use a maps API with the contact list to show all your contacts on a map, for example (see "Google APIs," later in this chapter).

Here are a few other mashups to get your brain juices pumping. All this stuff is included for you to use, and it's completely legal and free!

- ✓ **Geolocation and social networking:** Social networking is the "in" thing right now. Suppose you want to write an app that tweets your current location every 10 minutes throughout the day. You can, and it's easy. Use Android's location services and a third-party Twitter API (such as iTwitter), and you can do just that.
- ✓ **Geolocation and gaming:** Location-based gaming is gaining popularity. It's a great way

to really put your users into the game. A game might run a background service to check your current location and compare it with the locations of other users of the game in the same area. If another user is within 1 mile of you, for example, you could be notified, and you could challenge her to a battle. None of this would be possible without a strong platform such as Android and GPS technology.

✓ **Contacts and Internet:** With all these cool APIs at your disposal, it's easy to make full-featured apps by combining the functionality of two or more APIs. You can combine contacts and the Internet to create a greeting-card app, for example. Or you may just want to add an easy way for your users to contact you from an app or enable users to send the app to their friends. This is all possible with the built-in APIs.

The sky is the limit. All this cool functionality is literally in the palm of your hand. If you want to develop an app that records the geographic location of the device, you can with ease. Android really opens the possibilities by allow ing you to tap into these features easily. It's up to you, as the developer, to put them together in a way that can benefit your users.

Developers can do just about anything they want with Android, so be careful. Use your best judgment when creating and publishing apps for mass con sumption. Just because you want a live wallpaper that shows you doing the hula in your birthday suit doesn't mean that anyone else wants to see it.

Also, keep privacy laws in mind before you harvest your users' contact info for your own marketing scheme.

Android Programming Basics

You don't have to be a member of Mensa to program Android applications. I'm glad, because otherwise, I wouldn't be writing them! Programming for Android is simple because the default programming language of Android is Java. Although writing Android applications is fairly easy, programming in itself can be a difficult task to conquer.

If you've never programmed before, this book may not be the best place to start. I advise that you pick up a copy of *Beginning Programming with Java For Dummies,* by Barry Burd (Wiley Publishing), to learn the ropes. After you have a basic understanding of Java under your belt, you should be ready to tackle this book.

Although the majority of Android is Java, small parts of the framework aren't. Android also encompasses the XML language as well as basic Apache Ant scripting for build processes. I advise you to have a basic understanding of XML before delving into this book.

If you need an introduction to XML, check out *XML For Dummies,* by Lucinda Dykes and Ed Tittel (Wiley).

If you already know Java and XML, congratulations — you're ahead of the curve!

Java: Your Android programming language

Android applications are written in Java — not the full-blown Java that J2EE developers are used to, but a subset of Java that is sometimes known as the *Dalvik virtual machine.* This smaller subset of Java excludes classes that don't make sense for mobile devices. If you have any experience in Java, you should be right at home.

It may be a good idea to keep a Java reference book on hand, but in any case, you can always Google what you don't understand. Because Java is nothing new, you can find plenty of examples on the Web that demonstrate how to do just about anything.

In Java source code, not all libraries are included. Verify that the package is available to you. If it's not, an alternative is probably bundled with Android that can work for your needs.

Activities

Android applications are made up of one or more activities. Your app must contain at least one activity, but an Android application can contain several. Think of an activity as being a container for your UI, holding your UI as well as the code that runs it. It's kind of like a form, for you Windows programmers out there. I discuss activities in more detail in Chapters 3 and 5.

Intents

Intents make up the core message system that runs Android. An intent is composed of an action that it needs to perform (View, Edit, Dial, and so on) and data. The action is the general action to be performed when the intent is received, and the data is the data to operate on. The data might be a contact item, for example.

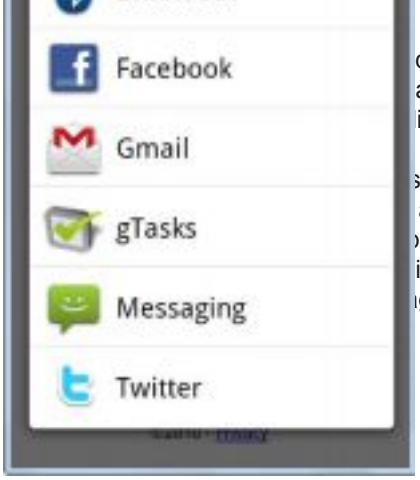
Intents are used to start activities and to communicate among various parts of the Android system. Your application can either broadcast an intent or receive an intent.

Sending messages with intents

When you broadcast an intent, you're sending a message telling Android to make something happen. This intent could tell Android to start a new activity from within your application, or it could start a different application.

Registering intent receivers

Just because you send a message doesn't mean that something will happen automatically. You have to register an *intent receiver* that listens for the intent and then tells Android what to do, whether the task is starting a new activity or starting a different app. If many receivers can accept a given



can be created to allow the user to pick the app she wants to ample is long-pressing an image in an image gallery. *Long* icking something for a long time to bring up a context menu.

s registered receivers handle the image-sharing intents. One and another is the messaging application (among various lications). Because you find more than one possible intent is presented with a chooser asking him what he should do: ging, or another application, as shown in Figure 1-1.

Figure 1-1:
A chooser.

If the Android system cannot find a match for the intent that was sent, and a chooser was not created manually, the application will crash due to a *run-time exception:* an unhandled error in the application. Android expects developers to know what they're doing. If you send an intent that a user's Android device doesn't know how to handle, the device crashes. It's best practice to create choosers for intents that don't target other activities within your application.

Cursorless controls

Unlike PCs, which let you use a mouse to move the cursor across the screen, Android devices let you use your fingers to do just about anything a mouse can do. But how do you right-click? Instead of supporting right-clicking, Android has implemented the long press. Press and hold a button, icon, or screen for an extended period of time, and a context menu appears. As a developer, you can create and manipulate context menus. You can allow users to use two fingers on an Android device instead of just one mouse cursor, for example. Keep in mind that fingers come in all sizes, however, and

design your user interface accordingly. Make the buttons large enough, with enough spacing, so that even users with large fingers can interact with your apps easily.

Views and widgets

What the heck is a view? A *view* is a basic UI element — a rectangular area on the screen that is responsible for drawing and event handling. I like to think of views as being basic controls, such as a label control in HTML. Here are a few examples of views:

- ✓ ContextMenu
- ✓ Menu
- ✓ View
- ✓ Surface view

Widgets are more-advanced UI elements, such as check boxes. Think of them as being the controls that your users interact with. Here are a few widgets:

- ✓ Button
- ✓ CheckBox
- ✓ DatePicker
- ✓ DigitalClock
- ✓ Gallery
- ✓ FrameLayout
- ✓ ImageView
- ✓ RelativeLayout
- ✓ PopupWindow

Many more widgets are ready for you to use. Check out the android.widget package in the Android documentation at http://developer.android.com/reference/android/widget/package-summary.html for complete details.

Asynchronous calls

Who called? I don't know anybody named Asynchronous, do you?

The AsyncTask class in Android allows you to run multiple operations at the same time without having to manage a separate thread yourself. AsyncTask not only lets you start a new process without having to clean up after your self, but also returns the result to the activity that started it. This allows you to have a clean programming model for asynchronous processing.

A *thread* is a process that runs separately from and simultaneously with every thing else that's happening.

When would you use asynchronous processing? I'm glad you asked! You'd use asynchronous processing for tasks that take a long time — network communication (Internet), media processing, or anything else that might make the user wait. If the user has to wait, you should use an asynchronous call and some type of UI element to let him know that something is happening.

Failing to use an asynchronous programming model can cause users of your application to believe that your application is buggy. Downloading the latest Twitter messages via the Internet takes time, for example. If the network gets slow, and you're not using an asynchronous model, the application will lock up, and the user will assume that something is wrong because the application doesn't respond to her interactions. If the application doesn't respond within a reasonable time that the Android OS defines, Android presents an "application not responding" (ANR) dialog box, as shown in Figure 1-2. At that time, the user can decide to wait or to close the application.

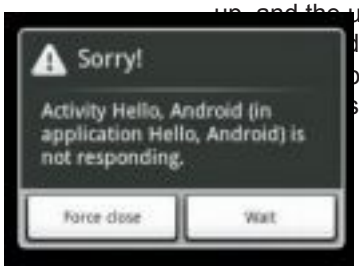


Figure 1-2: An ANR dialog box.

It's best practice to run CPU-expensive or long-running code inside another thread, as described in the Designing for Responsiveness page on the Android developer site (http://developer.android.com/guide/practices/design/responsiveness.html).

Background services

If you're a Windows user, you may already know what a *service* is: an application that runs in the background and doesn't necessarily have a UI. A classic example is an antivirus application that usually runs in the background as a service. Even though you don't see it, you know that it's running.

Most music players that can be downloaded from the Android Market run as background services. This is how you can listen to music while checking your e-mail or performing another task that requires the use of the screen.

Hardware Tools

Google exposes a plethora of functionality in Android, thus giving developers (even the independent guys) the tools needed to create top-notch, full featured mobile apps. Google has gone above and beyond by making it simple to tap into and make use of all the devices' available hardware.

To create a spectacular Android app, you should take advantage of all that the hardware has to offer. Don't get me wrong: If you have an idea for an app that doesn't need hardware assistance, that's okay too.

Android phones come with several hardware features that you can use to build your apps, as shown in Table 1-1.

Table 1-1 Android Device Hardware

Functionality Required	*Hardware*
Where am I?	GPS radio
Which way am I walking?	Built-in compass
Is my phone facing up or down?	Proximity sensor
Is my phone moving?	Accelerometer
Can I use my Bluetooth headphones?	Bluetooth radio
How do I record video?	Camera

Most Android phones are released with the hardware that I discuss in the following sections, but not all devices are created equal. Android is free for hardware manufacturers to distribute, so it's used in a wide range of devices, including some made by small manufacturers overseas (and it's not uncom mon for some of these phones to be missing a feature or two).

Also, as the technology advances, phone manufacturers are starting to add features that aren't yet natively supported by Android. But don't worry; man ufacturers that add hardware usually offer a software development kit (SDK) that lets developers tap into the device's unique feature. At this writing, HTC's Evo 4G, available from Sprint, is the only Android phone that comes

with a front-facing camera. Because this device is the first of its kind, Sprint has released an SDK that developers can use to access this cool new feature, as well as sample code that lets them implement the feature easily.

Android devices come in all shapes and sizes: phones, tablet computers, and e-book readers. You will find many other implementations of Android in the future, such as Google TV — an Android-powered home appliance — as well as cars with built-in Android-powered touchscreen computers. The engineers behind Android provide tools that let you easily deploy apps for multiple screen sizes and resolutions. Don't worry — the Android team has done all the hard

work for you. I cover the basics of screen sizes and densities in Chapter 4.

Touchscreen

Android phones have touchscreens, a fact that opens a ton of possibilities and can enhance users' interaction with your apps. Users can swipe, flip, drag, and pinch to zoom, for example, by moving a finger or fingers on the touchscreen. You can even use custom gestures for your app, which opens even more possibilities.

Android also supports *multitouch,* which means that the entire screen is touchable by more than one finger at a time.

Hardware buttons are old news. You can place buttons of any shape any where on the screen to create the UI that's best suited for your app.

GPS

The Android OS combined with a phone's GPS radio allows developers to access a user's location at any given moment. You can track a user's move ment as she changes locations. The Foursquare social-networking app is a good example; it uses GPS to determine the phone's location and then accesses the Web to determine which establishment or public place the user is in or near.

Another great example is the Maps application's ability to pinpoint your loca tion on a map and provide directions to your destination. Android combined with GPS hardware gives you access to the phone's exact GPS location. Many apps use this functionality to show you where the nearest gas station, coffee house, or even restroom is located. You can even use the maps API to pin point the user's current location on a map.

Accelerometer

Android comes packed with accelerometer support. An *accelerometer* is a device that measures acceleration. That sounds cool and all, but what can you do with it? If you want to know whether the phone is moving or being shaken, or even the direction in which it's being turned, the accelerometer can tell you.

You may be thinking, "Okay, but why do I care whether the phone is being shaken or turned?" Simple! You can use that input as a way to control your application. You can do simple things like determine whether the phone has been turned upside down and do something when it happens. Maybe you're making a dice game and want to immerse your users in the game play by having them shake the phone to roll the dice. This is the kind of functionality that is setting mobile devices apart from typical desktop personal computers.

SD Card

Android gives you the tools you need to access (save and load) files on the device's *SD Card* — a portable storage medium that you can insert into various phones and computers. If a device is equipped with an SD Card, you can use it to store and access files needed by your application. Android 2.2 allows you to install apps on the SD Card, but maybe your users have phones that don't get Android 2.2. Just because some users don't have the option of installing apps on the SD Card doesn't mean that you have to bloat your app with 20MB of resources and hog the phone's limited built-in memory. You can download some or all of your application's resources from your Web host and save them to the phone's SD Card. This makes your users happy and less likely to uninstall your app when space is needed.

Not all devices come with an SD Card installed, although most do. Always make sure that the user has an SD Card installed and that adequate space is available before trying to save files to it.

Software Tools

Various Android tools are at your disposal while writing Android applications. In the following sections, I outline some of the most popular tools that you will use in your day-to-day Android development process.

Internet

Thanks to the Internet capabilities of Android devices, real-time information is easy to obtain. As a user, you can use the Internet to see what time the next movie starts or when the next commuter train arrives. As a developer, you can use the Internet in your apps to access real-time, up-to-date data such as weather, news, and sports scores. You can also use the Web to store some of your application's assets, which is what Pandora and YouTube do.

Don't stop there. Why not offload some of your application's intense processes to a Web server when appropriate? This can save a lot of processing time in some cases and also helps keep your Android app streamlined. This arrangement is called *client–server computing* — a well-established software architecture in which the client makes a request to a server that is ready and willing to do something. The built-in Maps app is an example of a client accessing map and GPS data from a Web server.

Audio and video support

The Android OS makes including audio and video in your apps a breeze. Many standard audio and video formats are supported. Including multimedia content in your apps couldn't be any easier. Sound effects, instructional videos, background music, streaming video, and audio from the Internet can

all be added to your app with little to no pain. Be as creative as you want to be. The sky is the limit.

Contacts

Your app can access user contacts that are stored on the phone. You can use this feature to display the contacts in a new or different way. Maybe you don't like the built-in Contacts application. With the ability to access the con tacts stored on the phone, nothing is stopping you from writing your own. Maybe you write an app that couples the contacts with the GPS system and alerts the user when she is close to one of the contacts' addresses.

Use your imagination, but be responsible. You don't want to use contacts in a malicious way (see the next section).

Security

Android allows your apps to do a lot! Imagine if someone released an app that went through the contact list and sent the entire list to a server some where for malicious purposes. This is why most of the functions that modify the user's device or access its protected content need to have permissions to work. Suppose that you want to download an image from the Web and save it to the SD Card. To do so, you need to get permission to use the Internet so that you can download the file. You also need permission to save files to the SD Card. Upon installation of the application, the user is notified of the permissions that your app is requesting. At that point, the user can decide whether he wants to proceed with the installation. Asking for permission is as easy as implementing one line of code in your application's manifest file, which I cover in Chapter 3.

Google APIs

The Android OS isn't limited to making phone calls, organizing contacts, or installing apps. You have much more power at your fingertips. As a devel oper, you can integrate maps into your application. To do so, you have to use the maps APIs that contain the map widgets.

Pinpointing locations on a map

Perhaps you want to write an app that displays your current location to your friends. You could spend hundreds of hours developing a mapping system — or you could just use the Android Maps API. Google provides the Android Maps API, which you can use in your app, and just like everything else in Android, it's free! You can embed and use the API in your application to show your friends where you are; it won't take hundreds of hours or cost you a cent. Imagine all the juicy map goodness with none of the work developing it. Using the maps API, you can find just about anything with an address; the possibilities are endless. Display your friend's location, the nearest grocery store, or the nearest gas station — anything or anyplace with an address.

Getting around town with navigation

Showing your current location to your friends is cool, but wait — there's more! The Android Maps API can also access the Google Navigation API. Now you can pinpoint your location and also show your users how to get to that location.

Applications 23 The KISS principle

It's easy to overthink and overcomplicate things when developing applications. The hardest part is to remember the KISS (Keep It Simple, Stupid) principle. One way to overly complicate your code is to just dive in without understanding all the built-in APIs and knowing what they do. You can go that route, but doing so may take more time than just glossing over the Android documentation. You don't have to memorize it, but do yourself a favor and take a look at the documentation. You'll be glad you did when you see how easy it is to use the built-in functional ity and how much time it can save you. You can easily write multiple lines of code to do some thing that takes only one line. Changing the volume of the media player or creating a menu is a simple process, but if you don't know the APIs, you may end up rewriting them and in the end causing yourself problems.

When I started with my first app, I just dived in and wrote a bunch of code that managed the media player's volume. If I'd just looked into the Android documentation a little more, I'd have known that I could handle this with one line of code that's strategically placed inside my application. The same thing goes for the menu. I wrote a lot of code to create a menu, and if I'd only known that a menu framework already existed, it would have saved me several hours.

Another way to really muck things up is to add functionality that isn't needed. Most users want the easiest way to do things, so don't go making some fancy custom tab layout when a couple of menu items will suffice. Android comes with enough built-in controls (widgets) that you can use to accomplish just about anything. Using the built-in controls makes your app that much easier for your users to figure out because they already know and love these controls.

Messaging in the clouds

You may be thinking — clouds, I don't see a cloud in the sky! Well, I'm not talking about those kinds of clouds. The Android Cloud to Device Messaging framework allows you to send a notification from your Web server to your app. Suppose that you store your application's data in the cloud and down load all the assets the first time your app runs. But what if you realize after the fact that one of the images is incorrect? For the app to update the image, it needs to know that the image changed. You can send a cloud-to-device message (a message from the Web to the device) to your app, letting it know that it needs to update the image. This works even if your app is not running. When the device receives the message, it dispatches a message to start your app so that it can take the appropriate action.

Chapter 2

Prepping Your Development Headquarters

In This Chapter

- Becoming an Android application developer
- Collecting your tools of the trade
- Downloading and installing the Android SDK
- Getting and configuring Eclipse
- Working with the Android ADT

All the software that you need to develop Android applications is *free!*

That's where the beauty of developing Android applications lies. You'll be happy to find out that the basic building blocks you need to develop rich Android applications — the tools, the frameworks, and even the source code — are free. No, you don't get a free computer out of it, but you do get to set up your development environment and start developing applications for free, and you can't beat free. Well, maybe you can — such as someone paying you to

write an Android application, but you'll get there soon enough.

In this chapter, I walk you through the necessary steps to get the tools and frameworks installed so that you can start building kick-butt Android applications.

Developing the Android Developer Inside You

Becoming an Android developer isn't a complicated task. Actually, it's a lot simpler than you probably think. To see what's involved, ask yourself these questions:

- ✓ Do I want to develop Android applications?
- ✓ Do I like free software development tools?
- ✓ Do I like to pay no developer fees?
- ✓ Do I have a computer to develop on?

If you answered yes to all these questions, today is your lucky day; you're ready to become an Android developer. You may be wondering about the "no fees" part. Yep, you read that correctly: You pay no fees to develop Android applications.

There's always a catch, right? You can develop for free to your heart's content, but as soon as you want to publish your application to the Android Market — where you upload and publish your apps — you need to pay a small nominal registration fee. At this writing, the fee is $25.

Just to ease your mind about fees, it's important to note that if you're developing an application for a client, you can publish your application as a redistributable package that you can give him. Then your client can publish the application to the Android Market, using his Market account. This ensures that you don't have to pay a fee for client work — which means that you can be a bona fide Android developer and never have to pay a fee. Now, that's cool.

Assembling Your Toolkit

Now that you know you're ready to be an Android developer, grab your computer and get cracking on installing the tools and frameworks necessary to build your first blockbuster application.

Android source code

You should be aware that the full Android source code is open source, which means that it's not only free to use, but also free to modify. If you'd like to download the Android source code and create a new version of Android, you're free to do so. Check the Android Git repository. You can also download the source code at http://source.android.com.

Linux 2.6 kernel

Android was created on top of the open-source Linux 2.6 kernel. The Android team chose to use this kernel because it provided proven core features to develop the Android operating system on. The features of the Linux 2.6 kernel include (but aren't limited to) the following:

✓ **Security model:** The Linux kernel handles security between the application and the system.

✓ **Memory management:** The kernel handles memory management for you, leaving you free to develop your app.

✓ **Process management:** The Linux kernel manages processes well, allocating resources to processes as they need them.

✓ **Network stack:** The Linux kernel also handles network communication.

✓ **Driver model:** The goal of Linux is to ensure that everything works. Hardware manufacturers can build their drivers into the Linux build.

You can see a good sampling of the Linux 2.6 feature set in Figure 2-1.

Figure 2-1: Some of the Linux kernel features.

Android framework

Atop the Linux 2.6 kernel, the Android framework was developed with various features. These features were pulled from numerous open-source projects. The output of these projects resulted in the following:

✓ **The Android run time:** The Android run time is composed of Java core libraries and the Dalvik virtual machine.

✓ **Open GL (graphics library):** This cross-language, cross-platform application program interface (API) is used to produce 2D and 3D computer graphics.

✓ **WebKit:** This open-source Web browser engine provides the functionality to display Web content and simplify page loading.

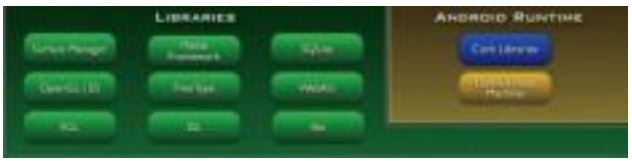
✓ **SQLite:** This open-source relational database engine is designed to be embedded in devices.

✓ **Media frameworks:** These libraries allow you to play and record audio and video.

✓ **Secure Sockets Layer (SSL):** These libraries are responsible for Internet security.

See Figure 2-2 for a list of common Android libraries.

Figure 2-2:



Android and other third-party libraries sit atop the Linux 2.6 kernel.

Application framework

You're probably thinking, "Well, that's all nice and well, but how do these libraries affect me as a developer?" It's simple: All these open-source frame works are available to you through Android. You don't have to worry about how Android interacts with SQLite and the surface manager; you just use them as tools in your Android tool belt. The Android team has built on a known set of proven libraries and has given them to you, all exposed through Android interfaces. These interfaces wrapped up the various libraries and made them useful to the Android platform as well as useful to you as a devel oper. Android has all these libraries built in the background and exposes these features to you without your having to build any of the functionality that they provide:

✓ **Activity manager:** Manages the activity life cycle.

✓ **Telephony manager:** Provides access to telephony services as well as some subscriber information, such as phone numbers.

✓ **View system:** Handles the views and layouts that make up your user interface (UI).

✓ **Location manager:** Finds out the device's geographic location.

Take a look at Figure 2-3 to see the libraries that make up the application

framework.

Figure 2-3:

A glimpse at part of the Android application framework.

From kernel to application, the Android operating system has been devel oped with proven open-source technologies. This allows you, as a developer, to build rich applications that have been fostered in the open-source com re of how the Android application frame

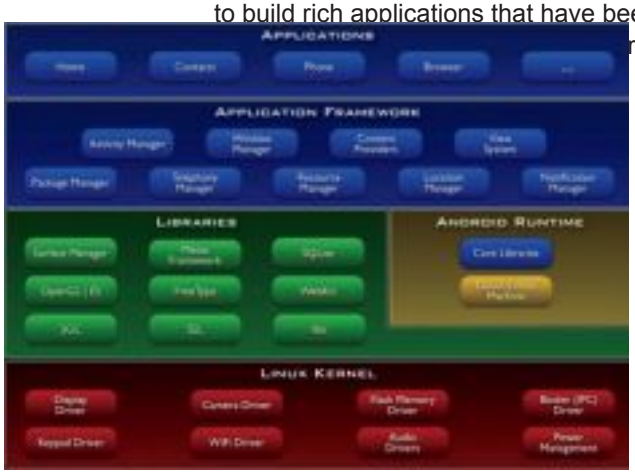


Figure 2-4: How the Android application framework stacks up. The Applications section is where your application sits.

Sometimes when developing an Android application, you'd like to use the same resource as in the core Android system. A good example would be an icon for a Settings menu option. By accessing the Android source code, you can browse the various resources and download the resources you need for your project. Having access to the source code also allows you to dig in and see exactly how Android does what it does.

Open Handset Alliance libraries

Huh? I didn't join any "alliance"; what's this about? Don't worry; you're not going to have to use the force to battle the unwieldy Darth Vader. It's not that big of a deal, and actually it's kind of cool. It's kind of like a bunch of really smart companies combining efforts to achieve the same goal.

The Open Handset Alliance (OHA) was announced in November 2007. At the time, the alliance consisted of 34 members, led by Google.

The OHA currently has 71 members. It's a group of technology and mobile companies that have come together to pursue innovation in the mobile field. The goal is to provide users comprehensive, compelling, and useful handsets. You can read more about the group at www.openhandsetalliance.com.

Now that's cool! The alliance has a lot of brilliant companies that are com bining their efforts to make the mobile world a better place. They include T-Mobile, Sprint, LG, Motorola, HTC, NVidia, and Texas Instruments.

You should be aware of the OHA because all the libraries that comprise the Android operating system (OS) are based on open-source code. Each member contributes in its own special way. Chip manufacturers ensure that chipsets support the platform; hardware manufacturers build devices; and other companies contribute intellectual property (code, documentation, and so on). The goal is to make Android a commercial success.

As these members contribute, they also start to innovate on the Android plat form. Some of this innovation makes it back into the Android source code, and some of it remains the intellectual property of the alliance members as decided by the OHA.

Just because one device has a fancy doohickey on it doesn't mean that another device will. The only thing that you can count on as a developer is the core Android framework. OHA members may have added an extra library to help facilitate something on a device, but there's no guarantee that this library will be available on another device in, say, Turkey or England. An exception occurs if you're developing for a particular device, and only that device, such as an e-book reader. If that hardware has the sole function of reading books, you can program it for just such a purpose. A real-world example of an e-book reader is the Barnes & Noble Nook, which is powered by Android. It has special Forward and Back buttons that other Android devices don't have. Therefore, you'd program for these buttons because this device is a special case (if you're developing for the Nook), but you can't expect these buttons to be used on other devices.

Java knowledge

The Java programming language is one of the glorious tools that make pro

gramming Android a breeze compared with programming for other mobile platforms. Whereas other languages insist that you manage memory, deal locate and allocate bytes, and then shift bits around like a game of dominoes, Java has a little buddy called the Java Virtual Machine (JVM) that helps take care of that for you. The JVM allows you to focus on writing code to solve a business problem by using a clean, understandable programming language (or to build that next really cool first-person shooter game you've been dreaming of) instead of focusing on the plumbing just to get the screens to show up.

You're expected to understand the basics of the Java programming language before you write your first Android application. If you're feeling a bit rusty and need a refresher course on Java, you can visit the Java tutorials site at http://java.sun.com/docs/books/tutorial.

I cover some Java in this book, but you may want to spend some time with a good book like *Java All-in-One For Dummies,* by Doug Lowe (Wiley), if you don't have any Java experience.

Tuning Up Your Hardware

You can develop Android applications on various operating systems, includ ing Windows, Linux, and Mac OS X. I do the development in this book on a Windows 7 operating system, but you can develop using Mac OS X or Linux instead.

Operating system

Android supports all the following platforms:

✓ Windows XP (32-bit), Vista (32- or 64-bit), and 7 (32- or 64-bit) ✓ Mac OS X 10.5.8 or later (x86 only)

✓ Linux (tested on Linux Ubuntu Hardy Heron)

Note that 64-bit distributions must be capable of running 32-bit applications.

Throughout the book, the examples use Windows 7 64-Bit Edition. Therefore, some of the screen shots may look a little different from what you see on your machine. If you're using a Mac or Linux machine, your paths may be different. Paths in this book look similar to this:

```
c:\path\to\file.txt
```

If you're on a Mac or Linux machine, however, your paths will look similar to this:

```
/path/to/file.txt
```

Computer hardware

Before you start installing the required software, make sure that your computer can run it adequately. I think it's safe to say that just about any desktop or laptop computer manufactured in the past four years will suffice. I wish I could be more exact, but I can't; the hardware requirements for Android simply weren't published when I wrote this book. The slowest computer that I have run Eclipse on is a laptop with a 1.6-GHz Pentium D processor with 1GB of RAM. I've run this same configuration under Windows XP and Windows 7, and both operating systems combined with that hardware can run and debug Eclipse applications with no problems.

To ensure that you can install all the tools and frameworks you'll need, make sure that you have enough disk space to accommodate them. The Android developer site has a list of hardware requirements, outlining how much hard drive space each component requires, at http://developer.android.com/sdk/requirements.html.

To save you time, I've compiled my own statistics from personal use of the tools and software development kits (SDKs). I've found that if you have about 3GB of free hard-drive space, you can install all the tools and frameworks necessary to develop Android applications.

Installing and Configuring Your Support Tools

Now it's starting to get exciting. It's time to get this Android going, but before you can do so, you need to install and configure a few tools, including SDKs:

- ✓ **Java JDK:** Lays the foundation for the Android SDK.
- ✓ **Android SDK:** Provides access to Android libraries and allows you to develop for Android.
- ✓ **Eclipse IDE (integrated development environment):** Brings together Java, the Android SDK, and the Android ADT (Android Development Tools), and provides tools for you to write your Android programs.
- ✓ **Android ADT:** Does a lot of the grunt work for you, such as creating the files and structure required for an Android app.

In the following sections, I show you how to acquire and install all these tools.

A benefit of working with open-source software is that most of the time, you can get the tools to develop the software for free. Android is no exception to that rule. All the tools that you need to develop rich Android applications are free of charge.

Getting the Java Development Kit

For some reason, the folks responsible for naming the Java SDK decided that it would be more appropriate to name it the Java Development Kit, or JDK for short.

Installing the JDK can be a somewhat daunting task, but I guide you through it one step at a time.

Downloading the JDK

Follow these steps to install the JDK:

1. Point your browser to http://java.sun.com/javase/downloads/index.jsp.

The Java SE downloads page appears.

:form (JDK) heading

dex.jsp page at this writing.

gh Software Update panel.

, asking you to specify which plat
ısing for your development work.

Figure 2-5:
Select JDK.

Choose JDK.

3. Using the Platform drop-down list, confirm your platform, and then click the Download button.

An optional Log in for Download screen appears.

s Step link at the bottom of the page.

ke to download the file.

a message box with a security warning, as shown in

alog box, select the location where you want to
d click Save.

Figure 2-6: The security warning.

The Web page shown in Figure 2-5 may look different in the future. To ensure that you're visiting the correct page, visit the Android SDK System Requirements page in the online Android documentation for a direct link to the Java SDK download page. View the requirements page at http://developer.android.com/sdk/requirements.html.

You must remember what version of the Java SDK you need to install. At this writing, Android 2.2 supports Java SDK versions 5 and 6. If you install the wrong version of Java, you'll get unexpected results during development.

Installing the JDK

When the download is complete, double-click the file to install the JDK. You are prompted by a dialog box that asks whether you want to allow the program to make changes to your computer. Click the Yes button. If you click the No button, the installation is stopped. When you're prompted to do so, read and accept the license agreement.

That's all there is to it! You have the JDK installed and are ready to move to the next phase. In this section, I show you how to install the Android SDK step by step.

Acquiring the Android SDK

The Android SDK is composed of a debugger, Android libraries, a device emulator, documentation, sample code, and tutorials. You can't develop Android apps without it.

Downloading the Android SDK

To download the Android SDK, follow these steps:

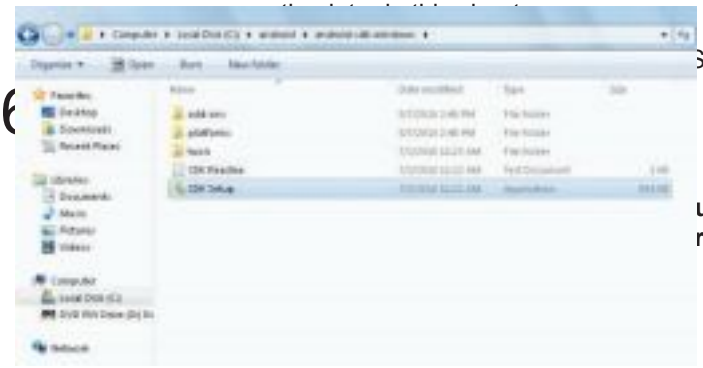
1. **Point your browser to http://developer.android.com/sdk/ index.html.**

2. **Choose the latest version of the SDK starter package for your platform.**

3. **Extract the SDK.**

I recommend extracting to c:\android because I reference this loca

SDK.



u extracted the SDK, and double
re 2-7.

Figure 2-7:
Double-click
SDK Setup.

5. If you're prompted to accept the authenticity of the file, click Yes.

ger dialog box opens.

l 2.2 check box.

lect version 2.2, as shown in Figure 2-8.
and greatest version of Android. You
the documentation and samples that
.2 (API 8).

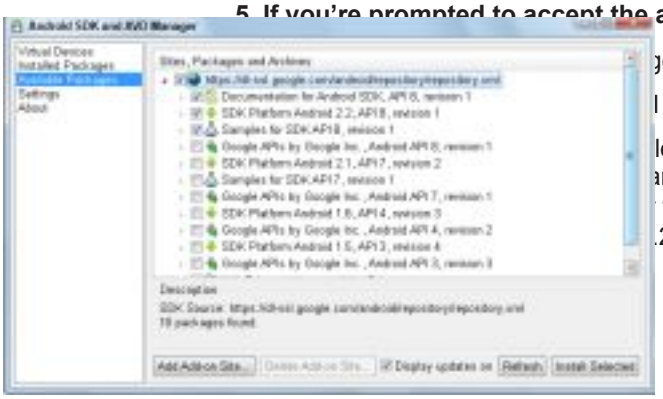


Figure 2-8:
Choose

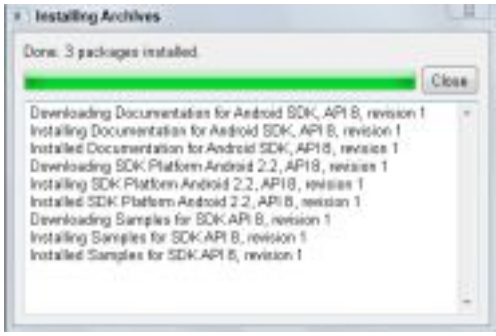
packages to install.

Every time a new version of the Android OS is released, Google also releases an SDK that contains access to the added functionality in that version. If you want to include Bluetooth functionality in your app, for example, make sure that you have Android SDK version 2.0 or later, because this functionality isn't available in earlier versions.

alog box opens.

o accept the license and then click

Figure 2-9: The Choose Packages to Install dialog box.



x, select Accept and click Install.

ives dialog box opens, displaying a progress bar (see

Figure 2-10: The Installing Archives dialog box.

10. When the archives installation is complete, click the Close button.

While the Android SDK is attempting to connect to the servers to obtain the files, you may occasionally receive a Failure to fetch URL error. If this happens to you, navigate to Settings, select Force https://... Sources

to be Fetched Using http://, and then attempt to download the available packages again.

Adding the Android NDK

The Android Native Development Kit (NDK) is a set of tools that allows you to embed compo nents that use native code — code that you've written in a native language such as C or C++.

If you decide to take on the NDK, you still have to download the SDK. The NDK isn't a replace ment for the SDK; it's an added functionality set that complements the SDK.

Following and setting your tools path

This step is optional, but I highly recommend setting the tools path because it saves you from having to remember and type the full path when you're accessing the Android Debug Bridge (adb) via the command line.

The adb lets you manage the state of an emulator or Android device so that you can debug your application or interact with the device at a high level. The adb tool is very in-depth, so I don't go into a lot of detail about it here; for detailed information, see the Android documentation.

To add the Android tools to your system-path variable, follow these steps:

1. **Open Control Panel, and double-click the System icon to open System Preferences.**

2. **Click the Advanced System Settings link (see Figure 2-11) to open the System ow.**

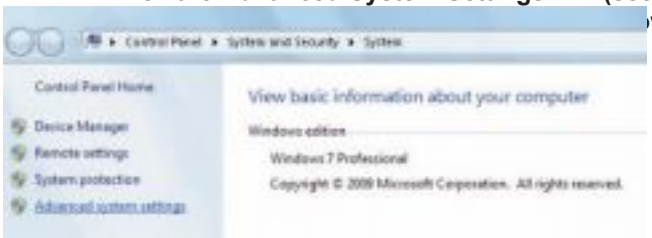


Figure 2-11: The Advanced System Settings link.

3. **Click the Environment Variables button (see Figure 2-12) to bring up the Environment Variables dialog box.**

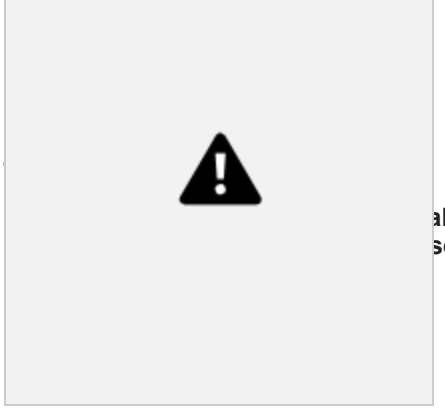
on (see Figure 2-13).

Figure 2-13: The Environment Variables window.

5. In the Variable Name field, type ANDROID.

6. Type the full path to the tools directory (c:\android\android-sdk windows\tools) in the Variable Value field (see Figure 2-14).

Figure 2-14: Setting up a new environment



ables window of the resulting dialog box select the PATH variable.

Figure 2-15: Editing the PATH variable.

9. Click Edit and then type the following text at the end of the Variable Value field:

;%ANDROID%

That's it; you're done. Now any time you access the Android tools directory, just use your newly created system variable.

In most operating systems, your system PATH variable won't be updated until you log out of and log back on to your operating system. If you find that your PATH variable values aren't present, try logging out of and logging back on to your computer.

Getting the Total Eclipse

Now that you have the SDK, you need an integrated development environment (IDE) to use it. It's time to download Eclipse!

Choosing the right Eclipse version

Downloading the correct version of Eclipse is very important. At this writing, Android doesn't support Eclipse Helios (version 3.6). Check the Android System Requirements page at http://developer.android.com/sdk/requirements.html. If you're still unsure, download Eclipse Galileo (ver

sion 3.5). When you download the file, you'll probably need to find the Older Versions link on the download page and select the latest Galileo version.

To download the correct version, navigate to the Eclipse downloads page (www.eclipse.org/downloads); select the Older Versions link; and then select Eclipse IDE for Java Developers. Eclipse IDE for JAVA EE Developers works as well.

Installing Eclipse

Eclipse is a self-contained executable file; after you unzip it, the program is installed. Even though you could stop here, it's best to pin a shortcut to your Start menu so that Eclipse is easy to find when you need it.

To install Eclipse, you need to extract the contents of the Eclipse .zip file to the location of your choice. For this example, I'll be using C:\Program Files\Eclipse.

To install Eclipse, follow these steps:

1. Double-click the shortcut that you just created to run Eclipse.

If you're running a recent version of Windows, the first time you run Eclipse, a Security Warning dialog box may appear, as shown in Figure

ox tells you that the publisher has not been verified
you still want to run the software. Clear the Always
This File check box, and click the Run button.

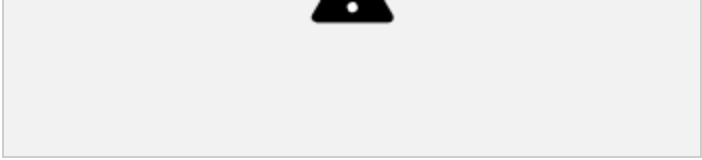
Figure 2-16: The Windows security warning.

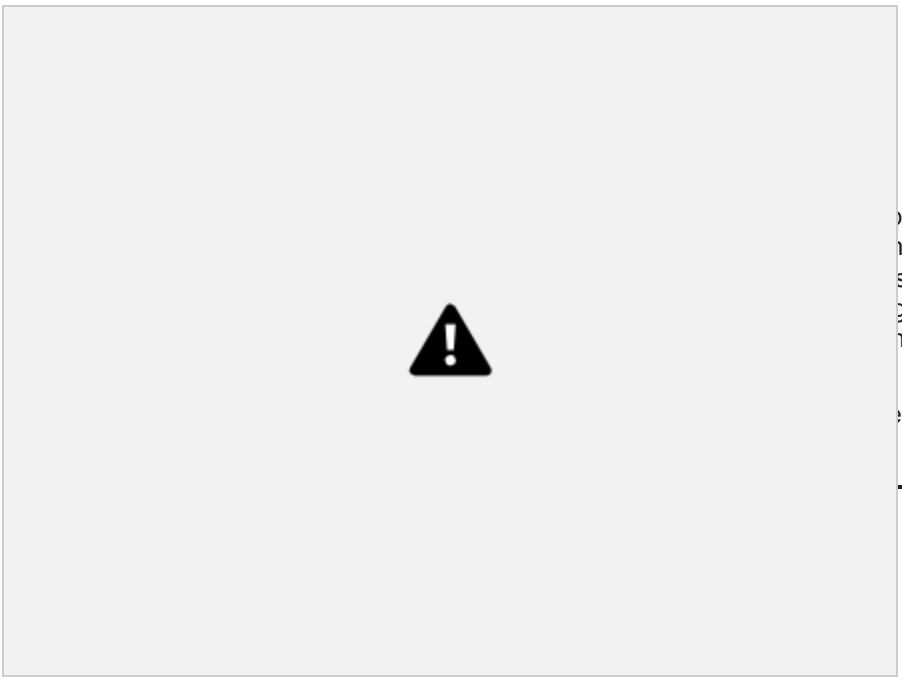
2. Set your workspace.

When Eclipse starts, the first thing you see is the Workspace Launcher dialog box, as shown in Figure 2-17. Here, you can modify your work space if you want, but for this book, I'm sticking with the default:

c:\users\<username>\workspace

Leave the Use This as the Default and Do Not Ask Again check box dese lected, and click the OK button.





a rate
e
s easy to
projects in
nen you have

screen,

Figure 2-18: The Eclipse welcome screen.

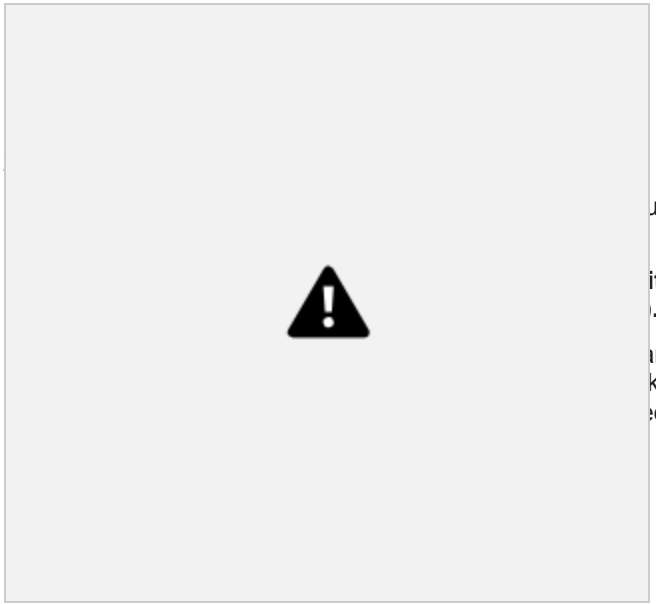
Click the arrow to go to the workbench.

3. Click the curved-arrow icon on the right side of the screen to go to the workbench.

Eclipse is installed and easily accessible. I show you how to add the Android Development Tools in the next section.

Configuring Eclipse

Android Development Tools (ADT) adds functionality to Eclipse to do a lot of the work for you. The ADT allows you to create new Android projects easily; it creates all the necessary base files so that you can start coding your appli cation quickly. It also allows you to debug your application using the Android SDK tools. Finally, it allows you to export a signed application file, known as an Android Package (APK), right from Eclipse, eliminating the need for some command-line tools. In the beginning, I had to use various command-line utilities to build an APK. Although that wasn't hard, it was tedious and some times frustrating. The ADT eliminates this frustrating process by guiding you through it "wizard style" from within Eclipse. I show you how to export a signed APK in Chapter 8.



steps:

ure 2-19). This window allows you to

ite that will display the Add
.

are is hosted on the
kes it easier for you to update the
ed.

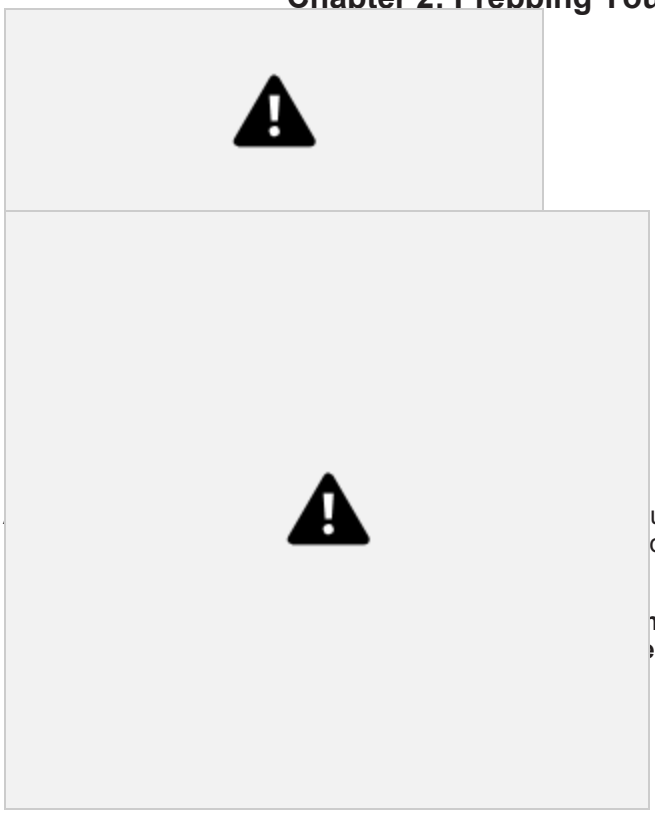
Figure 2-19: Click the Add button to add a

new site.

4. Type a name in the Name field.

I recommend using Android ADT, but it can be anything you choose.

5. Type https://dl-ssl.google.com/android/eclipse/ **in the Location field.**



u, and the available options are
dow of the

heck box next to Developer Tools,
e 2-21).

Figure 2-21: Select Developer Tools.

The Install Details dialog box should list both the Android Dalvik Debug Monitor Server (DDMS; see “Get physical with a real Android device,”

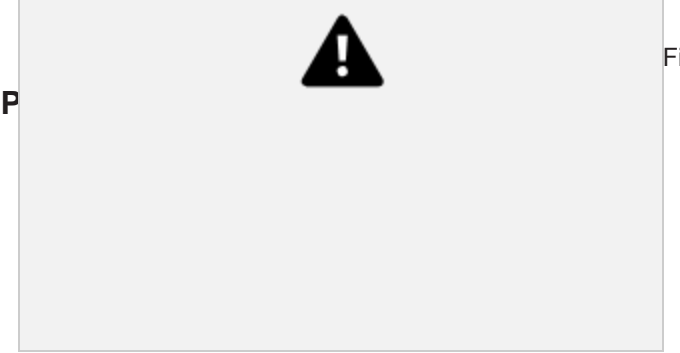


Figure 2-22).

Figure 2-22: DDMS and ADT listed in the Install Details dia log box.

8. **Click the Next button to review the software licenses.**
9. **Click the Finish button.**
10. **When you're prompted to do so, click the Restart Now button to restart Eclipse.**

    The ADT plug-in is installed.

Setting the location of the SDK

In this section, I guide you through the configuration process. I know that this seems like a lot to do, but you're almost done, and you have to do this work only once. Follow these steps:

1. **Choose Window➪Preferences.**

   The Preferences dialog box opens (see Figure 2-23).

2. **Select Android in the left pane.**
3. **Set the SDK Location to C:\android\android-sdk-windows.**
4. **Click OK.**

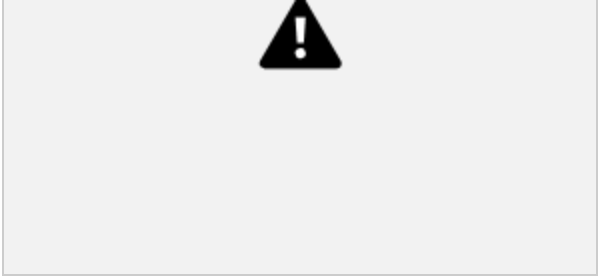


Figure 2-23: Specify the location of the SDK in the Preferences dialog box.

Eclipse is configured, and you're ready to start developing Android apps.

If you're having difficulty downloading the tools from https://dl-ssl.google.com/android/eclipse, try removing the *s* from https://, as follows: http://dl-ssl.google.com/android/eclipse.

Getting Acquainted with the Android Development Tools

Now that the tools of the trade are installed, I introduce you to the SDK and some of the tools that are included with it.

Navigating the Android SDK

Whoa! You find a lot of folders in the SDK! Don't worry; the folder structure of the Android SDK is pretty easy to understand when you get the hang of it. You need to understand the structure of the SDK for you to fully master it. Table 2-1 outlines what each folder is and what it contains.

Table 2-1 Folders in the Android SDK

SDK Folder	*Description*
usb_driver	Contains the drivers for Android devices. If you connect your Android device to the computer, you need to install this driver so that you can view, debug, and push applications to your phone via the ADT. The usb_driver folder won't be visible until you

install the USB driver.

tools Contains various tools that are available for use during development — debugging tools, view-management tools, and build tools, to name a few.

temp Provides a temporary swap for the SDK. At times, the SDK may need a temporary space to perform some work. This folder is where that work takes place.

samples Contains a bunch of sample projects for you to play with. Full source code is included.

platforms Contains the platforms that you target when you build Android applications, such as folders named android-8 (which is Android 2.2), android-4 (which is Android 1.6), and so on.

docs Contains a local copy of the Android SDK documentation.

add-ons Contains additional APIs that provide extra functionality. This folder is where the Google APIs reside; these APIs include mapping functionality. This folder remains empty until you install any of the Google Maps APIs.

Targeting Android platforms

Android platform is just a fancy way of saying *Android version.* At this writing, seven versions of Android are available, ranging from version 1.1 through version 2.2. You can target any platform that you choose.

Keep in mind that several versions of Android are still widely used on phones. If you want to reach the largest number of users, I suggest targeting an earlier version. If your app requires functionality that older platforms can't support, however, by all means target the new platform. It wouldn't make any sense to write a Bluetooth toggle widget targeting any platform earlier than 2.0 because earlier platforms can't use Bluetooth.

Figure 2-24 shows the percentage of each platform in use as of July 1, 2010. To view the current platform statistics, visit http://developer.android.com/resources/dashboard/platform-versions.html.

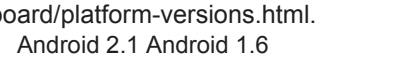


Figure 2-24: Android distribution as of July 1, 2010.

Android 2.2
Other

Using SDK tools for everyday development

You just installed the SDK tools. Now I introduce you to these tools so that they can work for you. The SDK tools are what you use to develop your Android apps. They allow you to develop applications easily as well as give you the ability to debug them. New features packed into every release enable you to develop for the latest version of Android.

Say hello to my little emulator

The emulator has to be my favorite tool of all. Not only does Google provide the tools you need to develop apps, but it also gives you this awesome little emulator that allows you to test your app! The emulator does have some limitations, however — it cannot emulate certain hardware components such as the accelerometer, but not to worry. Plenty of apps can be developed and tested using only an emulator.

When you're developing an app that uses Bluetooth, for example, you should use an actual device that has Bluetooth. If you develop on a speedy com puter, testing on an emulator is fast, but on slower machines, the emulator can take a long time to do a seemingly simple task. When I develop on an older machine, I usually use an actual device, but when I use my newer, faster machine, I typically use the emulator because I don't notice much lag, if any.

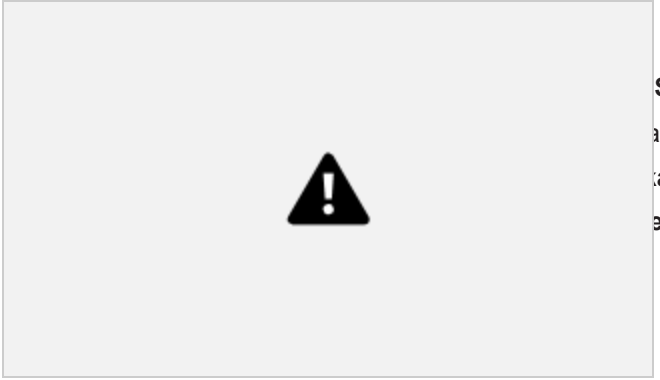
The emulator comes in handy for testing your app at different screen sizes and resolutions. It's not always practical or possible to have several devices connected to your computer at the same time, but you can run multiple emu lators with varying screen sizes and resolutions.

Get physical with a real Android device

The emulator is awesome, but sometimes you need an actual device to test on. The DDMS allows you to debug your app on an actual device, which comes in handy for developing apps that use hardware features that aren't or can't be emulated. Suppose that you're developing an app that tracks the user's location. You can send coordinates to the device manually, but at some point in your development, you probably want to test the app and find out whether it in fact displays the correct location. Using an actual device is the only way to do this.

If you develop on a Windows machine and want to test your app on a real device, you need to install a driver. If you're on a Mac or Linux machine, you can skip this section, because you don't need to install the USB driver.

To download the Windows USB driver for Android devices, follow these steps:



SDK and AVD Manager.

log box opens (see Figure 2-25).

ages.

elect the USB Driver package.

Figure 2-25: The available packages.

4. Click the Install Selected button.

The Choose Packages to Install dialog box opens.

ept the license and then click

ns, displaying a progress bar.

Figure 2-26: Click the Install button.

6. When the package finishes downloading and installing, click the Close button.

7. Exit the Android SDK and AVD Manager dialog box.

Debug your work

The DDMS equips you with the tools you need to find those pesky bugs, allowing you to go behind the scenes as your app is running to see the state of hardware such as wireless radios. But wait. There's more! It also simu lates actions that you normally need an actual device to do, such as sending

Global Positioning System (GPS) coordinates manually, simulating a phone call, or simulating a text message. I recommend getting all the DDMS details at http://developer.android.com/guide/developing/tools/ddms.html.

Try out the API and SDK samples

The API and SDK samples are provided to demonstrate how to use the functionality provided by the API and SDK. If you ever get stuck and can't figure out how to make something work, you should visit http://developer.android.com/resources/samples/index.html. Here, you can find samples of almost anything from using Bluetooth to making a two-way text application or a 2D game.

You also have a few samples in your Android SDK. Simply open the Android SDK and navigate to the samples directory, which contains various samples that range from interacting with services to manipulating local databases. You should spend some time playing with the samples. I've found that the best way to learn Android is to look at existing working code bases and then play with them in Eclipse.

Give the API demos a spin

The API demos inside the samples folder in the SDK are a collection of apps that demonstrate how to use the included APIs. Here, you can find sample apps with a ton of examples, such as these:

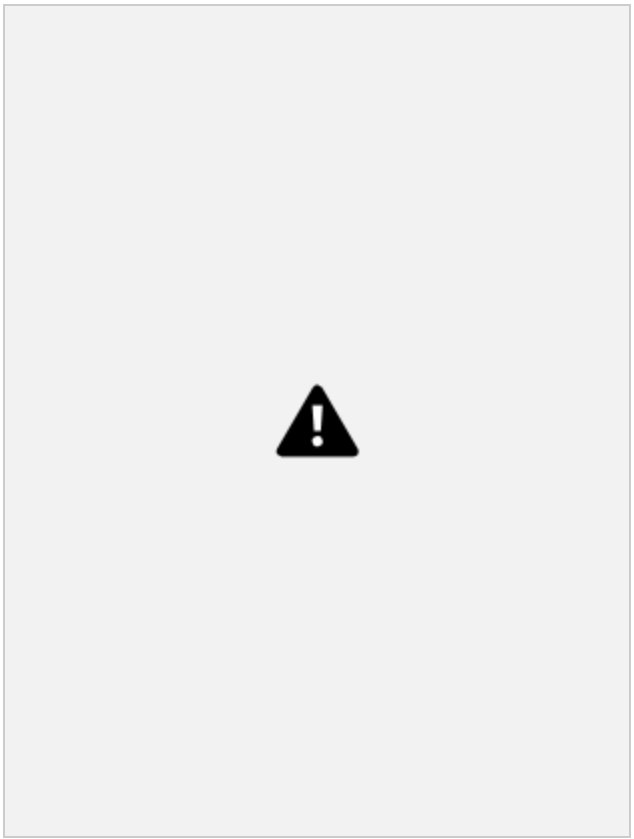
- ✓ Notifications
- ✓ Alarms
- ✓ Intents
- ✓ Menus
- ✓ Search
- ✓ Preferences
- ✓ Background services

If you get stuck or just want to prep yourself for writing your next spectacular Android application, check out the complete details at http://developer.android.com/resources/samples/ApiDemos/index.html.

Part II

Building and Publishing Your

First Android Application



In this part . . .

In Part II, I walk you through developing a useful Android application. I start with the basics of the Android tools and then delve into developing the screens and home screen widgets that users will interact with. When the application is complete, I demonstrate how to sign your application digitally so that you can deploy it to the Android Market. I finish the part with an in-depth view of publishing your application to the Android Market.

Chapter 3

Your First Android Project

In This Chapter

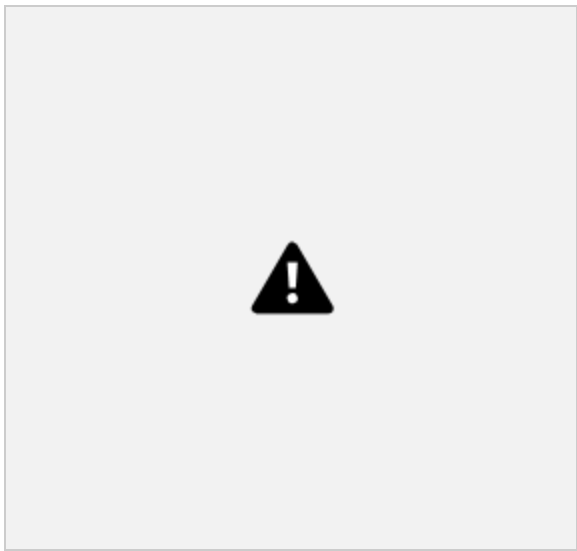
- Creating a new blank project in Eclipse
- Understanding errors
- Creating an emulator
- Setting up and copying launch configurations
- Running your first app
- Studying the anatomy of a project

You're excited to get started building the next best Android application known to man, right? Good! But before you create that next blockbuster application, I'm going to walk you through how to create your first Android application to help solidify a few key aspects in the Android project creation process. You will be creating a very simple "Hello Android" application that requires no coding whatsoever. What? No coding? How's that possible? Follow along; I'll show you.

Starting a New Project in Eclipse

First things first: You need to start Eclipse. After it's started, you should see something that looks similar to Figure 3-1. Now you're ready to start cooking with Android.

Remember setting up your development environment in the previous chap ter? I'm sure you do! In that chapter, you set up all the tools and frameworks necessary to develop Android applications, and in the process of doing so, the Eclipse Android Development Tools (ADT) plug-in was installed. The ADT plug-in gives you the power to generate new Android applications directly from within the Eclipse File menu. That's exactly what you're about to do; I think you're ready to create your first Android Application project. Follow these steps:



⇨**Project.**

dialog box opens, as shown in Figure 3-2.

Figure 3-2: The New Project/ Select a Wizard dialog box.

2. **From the New Project/Select a Wizard dialog box, expand the Android item by clicking the Android folder.**
3. **After the Android folder is expanded, click Android Project and then click the Next button.**

g box appears, as shown in Figure 3-3.

Hello Android**.**

ry important, the descriptive name that you
ct in the Eclipse workspace. After your proj
workspace is named with the project name

e default radio button Create New
check box Use Default Location

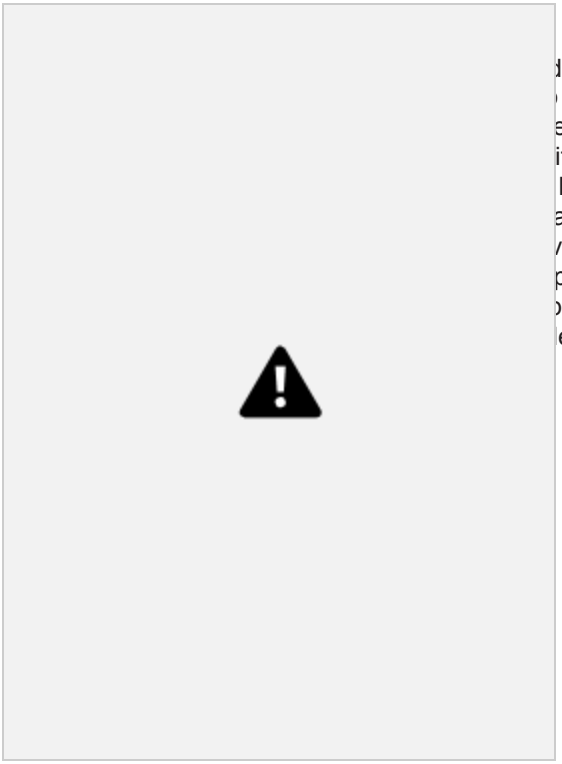
utomatically when a new project is cre
ntifies where the contents of your Eclipse
d in the file system. The contents are the
ndroid project.

Figure 3-3: The New Android Project dialog box.

When you set up Eclipse in Chapter 2, the Eclipse system asked you to set your default workspace. The workspace usually defaults to your home directory. A *home directory* is where the system places files pertinent to you. Figure 3-4 shows my home directory.

If you would rather store your files in a location other than the default workspace location, deselect the Use Default Location check box. This enables the Location text box. Click the Browse button, and select a location where you'd like your files to be stored.

6. In the Build Target section, select Android 2.2.



dentifies which application programming
develop under for this project. By selecting
ed to use the Android 2.2 framework. Doing
ith the Android 2.2 APIs, which include new
Manager and new speech-recognition APIs.
as the target, you would not be able to use
version 2.2 (or 2.1). Only the features in the
pported. If you installed other software devel
oter 2, you might have the option of selecting
lected version 1.6, you'd have access only to

Figure 3-4: My default workspace location for the Hello Android project is C:/Users/dfelker/workspace.