BAOU
Education for All

DR.BABASAHEB AMBEDKAR
# OPEN UNIVERSITY

BCA

BCAR-503
# Mobile Application Development

# MOBILE APPLICATION DEVELOPMENT (USING ANDROID)

DR. BABASAHEB AMBEDKAR OPEN UNIVERSITY

AHMEDABAD

**Editorial Panel**

**Author : Dr. Mahammad Idrish I. Sandhi**
**Associate Dean**
**Faculty of Computer Science**
**Sankalchand Patel University**
**Visnagar**

**Editor : Dr. Dharmeshkumar B. Bhavsar**
**Director**
**Chimanbhai Patel Institute of**
**Computer Applications**
**Affiliated to Gujarat University**

**Language Editor : Dr. Bhupesh Gupta**
**M.A., M.Phil., Ph. D.**
**Assistant Professor**
**Lokmanya College of Commerce**
**Ahmedabad**

**ISBN 978-93-91071-13-4**

**Edition : 2022**

**Acknowledgment**

Every attempt has been made to trace the copyright holders of material reproduced in this book. Should an infringement have occurred, we apologize for the same and will be pleased to make necessary correction/amendment in future edition of this book.

# ROLE OF SELF–INSTRUCTIONAL MATERIAL IN DISTANCE LEARNING

The need to plan effective instruction is imperative for a successful distance teaching repertoire. This is due to the fact that the instructional designer, the tutor, the author (s) and the student are often separated by distance and may never meet in person. This is an increasingly common scenario in distance education instruction. As much as possible, teaching by distance should stimulate the student's intellectual involvement and contain all the necessary learning instructional activities that are capable of guiding the student through the course objectives. Therefore, the course / self–instructional material is completely equipped with everything that the syllabus prescribes.

To ensure effective instruction, a number of instructional design ideas are used and these help students to acquire knowledge, intellectual skills, motor skills and necessary attitudinal changes. In this respect, students' assessment and course evaluation are incorporated in the text.

The nature of instructional activities used in distance education self–instructional materials depends on the domain of learning that they reinforce in the text, that is, the cognitive, psychomotor and affective. These are further interpreted in the acquisition of knowledge, intellectual skills and motor skills. Students may be encouraged to gain, apply and communicate (orally or in writing) the knowledge acquired. Intellectual–skills objectives may be met by designing instructions that make use of students' prior knowledge and experiences in the discourse as the foundation on which newly acquired knowledge is built.

The provision of exercises in the form of assignments, projects and tutorial feedback is necessary. Instructional activities that teach motor skills need to be graphically demonstrated and the correct practices provided during tutorials. Instructional activities for inculcating change in attitude and behaviour should create interest and demonstrate need and benefits gained by adopting the required change. Information on the adoption and procedures for practice of new attitudes may then be introduced.

Teaching and learning at a distance eliminate interactive communication cues, such as pauses, intonation and gestures, associated with the face–to–face method of teaching. This is particularly so with the exclusive use of print media. Instructional activities built into the instructional repertoire provide this missing interaction between the student and the teacher. Therefore, the use of instructional activities to affect better distance teaching is not optional, but mandatory.

Our team of successful writers and authors has tried to reduce this.

Divide and to bring this Self–Instructional Material as the best teaching and communication tool. Instructional activities are varied in order to assess the different facets of the domains of learning.

Distance education teaching repertoire involves extensive use of self–instructional materials, be they print or otherwise. These materials are designed to achieve certain pre–determined learning outcomes, namely goals and objectives that are contained in an instructional plan. Since the teaching process is affected over a distance, there is need to ensure that students actively participate in their learning by performing specific tasks that help them to understand the relevant concepts. Therefore, a set of exercises is built into the teaching repertoire in order to link what students and tutors do in the framework of the course outline. These could be in the form of students' assignments, a research project or a science practical exercise. Examples of instructional activities in distance education are too numerous to list. Instructional activities, when used in this context, help to motivate students, guide and measure students' performance (continuous assessment)

# PREFACE

We have put in lots of hard work to make this book as user–friendly as possible, but we have not sacrificed quality. Experts were involved in preparing the materials. However, concepts are explained in easy language for you. We have included many tables and examples for easy understanding.

We sincerely hope this book will help you in every way you expect.

All the best for your studies from our team!

# MOBILE APPLICATION DEVELOPMENT (USING ANDROID)

**Contents**

Version, Specifying the Target SDK Version, Specifying the Maximum SDK Version, Enforcing Application System Requirements Working with Permissions, Specifying Supported Input Methods, Specifying Required Device Features, Specifying Supported Screen Sizes, Working with External Libraries, Working with Permissions, Registering Permissions Your Application Requires, Registering Permissions Your Application Grants to Other Applications

**Unit 7 MANAGING APPLICATION RESOURCES IN A HIERARCHY & WORKING WITH DIFFERENT TYPES OF RESOURCES**

Introduction, What Are Resources ?, Storing Application Resources, Understanding the Resource Directory Hierarchy, Using the Android Asset Packaging Tool, Resource Value Types, Storing Different Resource Value Types, Storing Simple Resource Types Such as Strings, Storing Graphics, Animations, Menus, and Files, Understanding How Resources Are Resolved, Accessing Resources Programmatically, Working with different types of resources, Working with String Resources, Working with String Arrays, Working with Boolean Resources, Working with Integer Resources, Working with Colors, Working with Dimensions, Working with Simple Drawables, Working with Images, Working with Animation, Working with Menus, Working with XML Files, Working with Raw Files, Working with Layouts, Working with Styles

## BLOCK 3 : ANDROID USER INTERFACE DESIGN ESSENTIALS

**Unit 8 USER INTERFACE SCREEN ELEMENTS**

Introduction, Introducing the Android Control, Introducing the Android Layout, Displaying Text to Users with TextView, Configuring Layout and Sizing, Creating Contextual Links in Text, Retrieving Text Input Using EditText Controls, Giving Users Input Choices Using Spinner Controls, Using Buttons, Check Boxes, and Radio Groups, Using Basic Buttons, Using Check Boxes and Toggle Buttons, Using Radio Groups and

Radio Buttons, Getting Dates and Times From Users, Using Indicators to Display Data to Users, Indicating Progress with Progress Bar, Adjusting Progress with Seek Bar, Displaying Rating Data with Rating Bar, Showing Time Passage with the Chronometer, Displaying the Time, Providing Users with Options and Context Menus, Enabling the Options Menu, Enabling the Context Menu, Handling User Events, Listening for Touch Mode Changes, Listening for Events on the Entire Screen, Listening for Long Clicks, Listening for Focus Changes Working with Dialogs, Exploring the Different Types of Dialogs, Tracing the Lifecycle of a Dialog, Working with Styles, Working with Themes

**Unit 9 DESIGNING USER INTERFACES WITH LAYOUTS**

Introduction, Creating User Interfaces in Android, Creating Layouts Using XML Resources, Creating Layouts Programmatically, Organizing Your User Interface, Understanding View Versus ViewGroup, Sub Topic, Using Built–In Layout Classes, Using FrameLayout, Using LinearLayout, Using RelativeLayout, Using TableLayout

**Unit 10 DRAWING AND WORKING WITH ANIMATION**

Introduction, Drawing on the Screen, Working with Canvases and Paints, Working with Text, Working with Bitmaps, Working with Shapes, Working with Animation, Working with Frame–by–Frame Animation, Working with Tweened Animations

---

**BLOCK 4 : COMMON ANDROID APIS & DEPLOYING ANDROID APPLICATION**

---

**Unit 11 MANAGING DATA USING SQLITE**

Introduction, Creating a SQLite Database, Creating a SQLite Database Instance Using the Application Context, Finding the Application's Database File on the Device File System, Configuring the SQLite Database Properties, Creating Tables and Other SQLite Schema Objects, Creating, Updating, and Deleting Database Records, Inserting Records, Updating Records, Deleting Records, Working with Transactions, Querying SQLite Databases, Working with Cursors, Executing Simple Queries, Closing and Deleting a SQLite Database, Deleting Tables and Other SQLite Objects, Closing a SQLite Database, Deleting a SQLite Database Instance Using the Application Context, Designing Persistent Databases, Keeping Track of Database Field Names, Extending the SQLiteOpenHelper Class, Binding Data to the Application User Interface, Working with Database Data Like Any Other Data, Binding Data to Controls Using Data Adapters

**Unit 12** **USING ANDROID NETWORKING APIS**

Introduction, Accessing the Internet (HTTP), Reading Data from the Web, Using HttpURLConnection, Parsing XML from the Network, Processing Asynchronously, Working with AsyncTask, Using Threads for Network Calls, Displaying Images from a Network Resource, Retrieving Android Network Status

**Unit 13** **USING ANDROID WEB APIS & TELEPHONY APIS**

Introduction, Browsing the Web with WebView, Designing a Layout with a WebView Control, Loading Content into a WebView Control, Adding Features to the WebView Control, Building Web Extensions Using WebKit, Browsing the WebKit APIs, Extending Web Application Functionality to Android, Using Android Telephony APIs, Working with Telephony Utilities, Using SMS, Making and Receiving Phone Calls

**Unit 14** **SELLING YOUR ANDROID APPLICATION**

Introduction, Choosing the Right Distribution Model, Packaging Your Application for Publication, Preparing Your Code to Package, Packing and Signing Your Application, Distributing Your Applications, Selling Your Application on the Android Market, Signing Up for a Developer Account on the Android Market

Dr. Babasaheb Ambedkar Open University Ahmedabad

BCAR-503

# MOBILE APPLICATION DEVELOPMENT (USING ANDROID)

## BLOCK 1 : INTRODUCTION TO ANDROID

# INTRODUCTION TO ANDROID

## Block Introduction :

Pioneered by the Open Handset Alliance and Google, Android is a hot, young, free, open–source mobile platform making waves in the wireless world. This book provides comprehensive guidance for software development teams on designing, developing, testing, debugging, and distributing professional Android applications. If you're a veteran mobile developer, you can find tips and tricks to streamline the development process and take advantage of Android's unique features. If you're new to mobile development, this book provides everything you need to make a smooth transition from traditional software development to mobile development–specifically, its most promising new platform : Android

## Block Objectives :

**After learning this block, you will be able to understand :**

- The history of mobile application development
- Various mobile proprietary platforms available in market
- Formation of OHA (Open Handset Alliance)
- Android Platform architecture
- Various versions and Codename of Android
- To develop the mobile application using Android
- To understand the basic of Android Application Development

## Block Structure :

Unit 01

# HISTORY OF MOBILE SOFTWARE DEVELOPMENT

UNIT STRUCTURE

## 1.0 Learning Objectives :

**After learning this unit, you will be able to understand :**

- The history of mobile application development
- Various mobile proprietary platforms available in market

## 1.1 Introduction :

To understand what makes Android so compelling, we must examine how mobile development has evolved and how Android differs from competing platforms. The mobile development community is at a tipping point. Mobile users demand more choice, more opportunities to customize their phones, and more functionality. Mobile operators want to provide value–added content to their subscribers in a manageable and lucrative way. Mobile developers want the freedom to develop the powerful mobile applications users demand with minimal roadblocks to success. Finally, handset manufacturers want a stable, secure, and affordable platform to power their devices. Up until now a single mobile platform has adequately addressed the needs of all the parties. Enter Android, which is a potential game–changer for the mobile development community. An innovative and open platform, Android is well positioned to address the growing needs of the mobile marketplace. This chapter explains what Android is, how and why it was developed, and where the platform fits in to the established mobile marketplace.

### 1.1.1 The Way Back When :

Remember way back when a phone was just a phone ? When we relied on fixed landlines ? When we ran for the phone instead of pulling it out of our pocket ? When we lost our friends at a crowded ballgame and waited around for hours hoping to reunite ? When we forgot the grocery list (see Figure 1.1) and had to find a payphone or drive back home again ?

**Figure 1.1 : Mobile Phone has become a crucial shopping accessory**

Those days are long gone. Today, commonplace problems such as these are easily solved with a one–button speed dial or a simple text message like "WRU ?" or "20 ?" or "Milk and ?"

Our mobile phones keep us safe and connected. Now we roam around freely, relying on our phones not only to keep in touch with friends, family, and coworkers, but also to tell us where to go, what to do, and how to do it. Even the most domestic of events seem to revolve around my mobile phone.

My point here ? Mobile phones can solve just about anything–and we rely on them for everything these days. You notice that I used half a dozen different mobile applications over the course of this story. Each application was developed by a different company and had a different user interface. Some were well designed; others not so much. I paid for some of the applications, and others came on my phone. As a user, I found the experience functional, but not terribly inspiring. As a mobile developer, I wished for an opportunity to create a more seamless and powerful application that could handle all I'd done and more. I wanted to build a better bat trap, if you will. Before Android, mobile developers faced many roadblocks when it came to writing applications. Building the better application, the unique application, the competing application, the hybrid application, and incorporating many common tasks such as messaging and calling in a familiar way were often unrealistic goals. To understand why, let's take a brief look at the history of mobile software development.

**1.1.2 "The Brick" :**

The Motorola DynaTAC 8000X was the first commercially available cell phone. First marketed in 1983, it was 13 × 1.75 × 3.5 inches in dimension, weighed about 2.5 pounds, and allowed you to talk for a little more than half an hour. It retailed for $3,995, plus hefty monthly service fees and per–minute charges. We called it "The Brick," and the nickname stuck for many of those early mobile phones we alternatively loved and hated. About the size of a brick, with a battery power just long enough for half a conversation, these early mobile handsets were mostly seen in the hands of traveling business execs, security personnel, and the wealthy. First–generation mobile phones were just too expensive. The service charges alone would bankrupt the average person, especially when

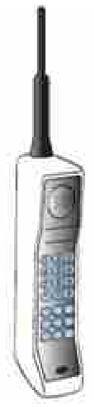


**Figure 1.2 : The first commercially available mobile phone : The Motorola DynaTAC**

roaming. Early mobile phones were not particularly full featured. (Although, even the Motorola DynaTAC, shown in Figure 1.2, had many of the buttons we've come to know well, such as the SEND, END, and CLR buttons.) These early phones did little more than make and receive calls and, if you were lucky, there was a simple contacts application that wasn't impossible to use.

The first–generation mobile phones were designed and developed by the handset manufacturers. Competition was fierce and trade secrets were closely guarded. Manufacturers didn't want to expose the internal workings of their handsets, so they usually developed the phone software in–house. As a developer, if you weren't part of this inner circle, you had no opportunity to write applications for the phones. It was during this period that we saw the first "time–waster" games begin to appear. Nokia was famous for putting the 1970s video game Snake on some of its earliest monochrome phones. Other manufacturers followed suit, adding games such as Pong, Tetris, and Tic–Tac–Toe. These early phones were flawed, but they did something important–they changed the way people thought about communication. As mobile phone prices dropped, batteries improved, and reception areas grew, more and more people began carrying these handy devices. Soon mobile phones were more than just a novelty. Customers began pushing for more features and more games. But there was a problem. The handset manufacturers didn't have the motivation or the resources to build every application users wanted. They needed some way to provide a portal for entertainment and information services without allowing direct access to the handset.

## 1.2 Wireless Application Protocol (WAP) :

As it turned out, allowing direct phone access to the Internet didn't scale well for mobile. By this time, professional websites were full color and chock full of text, images, and other sorts of media. These sites relied on JavaScript, Flash, and other technologies to enhance the user experience, and they were often designed with a target resolution of 800x600 pixels and higher. When the first clamshell phone, the Motorola StarTAC, was released in 1996, it merely had an LCD 10–digit segmented display. (Later models would add a dot–matrix type display.) Meanwhile, Nokia released one of the first slider phones, the 8110–fondly referred to as "The Matrix Phone" because the phone was heavily used in films. The 8110 could display four lines of text with 13 characters per line. Figure 1.3 shows some of the common phone form factors.

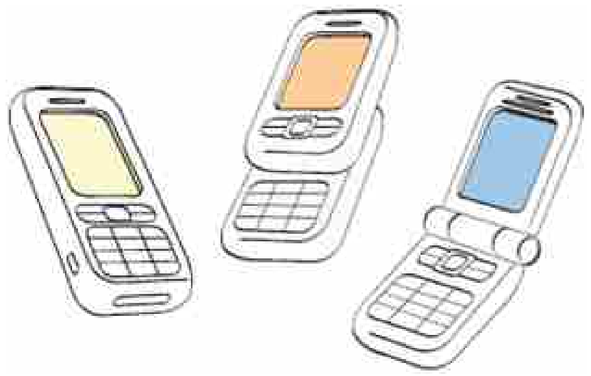


**Figure 1.3 : Various mobile phone form factors : the candy bar, the slider, and the clamshell.**

With their postage stamp–sized low–resolution screens and limited storage and processing power, these phones couldn't handle the data–intensive operations required by traditional web browsers. The bandwidth requirements for data transmission were also costly to the user. The Wireless Application Protocol (WAP) standard emerged to address these concerns. Simply put, WAP was a stripped–down version of HTTP, which is the backbone protocol of the Internet. Unlike traditional web browsers browsers were designed to run within the memory and bandwidth constraints of the phone. Third–party WAP sites served up pages written in a markup language called Wireless Markup Language (WML). These pages were then displayed on the phone's WAP browser. Users navigated as they would on the Web, but the pages were much simpler in design. The WAP solution was great for handset manufacturers. The pressure was off– they could write one WAP browser to ship with the handset and rely on developers to come up with the content users wanted. The WAP solution was great for mobile operators. They could provide a custom WAP portal, directing their subscribers to the content they wanted to provide, and rake in the data charges associated with browsing, which were often high. Developers and content providers didn't deliver. For the first time, developers had a chance to develop content for phone users, and some did so, with limited success. Most of the early WAP sites were extensions of popular branded websites, such as CNN.com and ESPN.com, which were looking for new ways to extend their readership. Suddenly phone users accessed the news, stock market quotes, and sports scores on their phones. Commercializing WAP applications was difficult, and there was no built–in billing mechanism. Some of the most popular commercial WAP applications that emerged during this time were simple wallpaper and ringtone catalogues that enabled users to personalize their phones for the first time. For example, a user browsed a WAP site and requested a specific item. He filled out a simple order form with his phone number and his handset model. It was up to the content provider to deliver an image or audio file compatible with the given phone. Payment and verification were handled through various premium priced delivery mechanisms such as Short Message Service (SMS), Enhanced Messaging Service (EMS), Multimedia Messaging Service (MMS), and WAP Push. WAP browsers, especially in the early days, were slow and frustrating. Typing long URLs with the numeric keypad was onerous pages were often difficult to navigate. Most WAP sites were written one time for all phones and did not account for individual phone specifications. It didn't matter if the end user's phone had a big color screen or a postage stamp–sized monochrome screen; the developer couldn't tailor the user's experience. The result was a mediocre and not very compelling experience for everyone involved. Content providers often didn't bother with a WAP site and instead just advertised SMS short codes on TV and in magazines. In this case, the user sent a premium SMS message with a request for a specific wallpaper or ringtone, and the content provider sent it back. Mobile operators generally liked these delivery mechanisms because they received a large portion of each messaging fee. WAP fell short of commercial expectations. In some markets, such as Japan, it flourished, whereas in others, such as the United States, it failed to take off. Handset screens were too small for surfing. Reading a sentence fragment at a time, and then waiting seconds for the next segment to download, ruined the user experience, especially because every second of downloading was often charged to the user. Critics began to call WAP "Wait and Pay."

Finally, the mobile operators who provided the WAP portal (the default home page loaded when you started your WAP browser) often restricted which WAP sites were accessible. The portal enabled the operator to restrict the number of sites users could browse and to funnel subscribers to the operator's preferred content providers and exclude competing sites. This kind of walled garden approach further discouraged third–party developers, who already faced difficulties in monetizing applications.

## 1.3 Proprietary Mobile Platforms :

It came as no surprise that users wanted more–they will always want more. Writing robust applications with WAP, such as graphic–intensive video games, was nearly impossible. The 18–year–old to 25–year–old sweet–spot demographic–the kids with the disposable income most likely to personalize their phones with wallpapers and ringtones–looked at their portable gaming systems and asked for a device that was both a phone and a gaming device or a phone and a music player. They argued that if devices such as Nintendo's Game Boy could provide hours of entertainment with only five buttons, why not just add phone capabilities ? Others looked to their digital cameras, Palms, Blackberries, iPods, and even their laptops and asked the same question. The market seemed to be teetering on the edge of device convergence. Memory was getting cheaper, batteries were getting better, and PDAs and other embedded devices were beginning to run compact versions of common operating systems such as Linux and Windows. The traditional desktop application developer was suddenly a player in the embedded device market, especially with smartphone technologies such as Windows Mobile, which they found familiar. Handset manufacturers realized that if they wanted to continue to sell traditional handsets, they needed to change their protectionist policies pertaining to handset design and expose their internal frameworks to some extent. A variety of different proprietary platforms emerged–and developers are still actively creating applications for them. Some smartphone devices ran Palm OS (now Garnet OS) and RIM BlackBerry OS. Sun Microsystems took its popular Java platform and J2ME emerged (now known as Java Micro Edition [Java ME]). Chipset maker Qualcomm developed and licensed its Binary Runtime Environment for Wireless (BREW). Other platforms, such as Symbian OS, were developed by handset manufacturers such as Nokia, Sony Ericsson, Motorola, and Samsung. The Apple iPhone OS (OS X iPhone) joined the ranks in 2008. Figure 1.4 shows several different phones, all of which have different development platforms. Many of these platforms have associated developer programs. These programs keep the developer communities small, vetted, and under contractual agreements on what they can and cannot do. These programs are often required and developers must pay for them. Each platform has benefits and drawbacks. Of course, developers love to debate about which platform is "the best." (Hint : It's usually the platform we're currently developing for.) The truth is that no one platform has emerged victorious. Some platforms are best suited for commercializing games and making millions–if your company has brand backed. Other platforms are more open and suitable for the hobbyist or vertical market applications. No mobile platform is best suited for all possible applications. As a result, the mobile phone has become increasingly fragmented, with all platforms sharing part of the pie.

**Figure 1.4 : Phones from various mobile device platforms.**

For manufacturers and mobile operators, handset product lines quickly became complicated. Platform market penetration varies greatly by region and user demographic. Instead of choosing just one platform, manufacturers and operators have been forced to sell phones for all the different platforms to compete in the market. We've even seen some handsets supporting multiple platforms. (For instance, Symbian phones often also support J2ME.) The mobile developer community has become as fragmented as the market. It's nearly impossible to keep track of all the changes in the market. Developer specialty niches have formed. The platform development requirements vary greatly. Mobile software developers work with distinctly different programming environments, different tools, and different programming languages. Porting among the platforms is often costly and not straightforward. Keeping track of handset configurations and testing requirements, signing and certification programs, carrier relationships, and application marketplaces have become complex spin–off businesses of their own.

It's a nightmare for the ACME Company that wants a mobile application. Should it develop a J2ME application ? BREW ? iPhone ? Windows Mobile ? Everyone has a different kind of phone. ACME is forced to choose one or, worse, all of the platforms. Some platforms allow for free applications, whereas others do not. Vertical market application opportunities are limited and expensive. As a result, many wonderful applications have not reached their desired users, and many other great ideas have not been developed at all.

❑ **Check Your Progress :**

1. What is the name of first commercial cell phone available in market ?

   (A) DynaTec (B) Binary Phone

   (C) Motorola DynaTAC 8000X (D) Slider

2. WAP stans for ________

   (A) Wire Applied Panel (B) Wireless Application Protocol

   (C) Without Approve Package (D) Wireless Applied Protocol

3. J2EE Stands for ________

   (A) Java 2 Enterprise Edition (B) Java 2 Entertainment Edition

   (C) Java 2 Edition Enterprise (D) Java 2 Editor Edition

4. Nokia released one of the first slider phones, the 8110–fondly referred to as ________

   (A) "The Matrix Phone" (B) "The Phone"

   (C) "The Phoenix" (D) "Slider Phone"

5. BREW Stands for ________

   (A) Binary Runway Environment for Wireless

   (B) Binary Rough Entertainment for Wireless

   (C) Binary Runtime Environment for Wireless

   (D) Binary Routine Environment for Wireless

## 1.4 Let Us Sum Up :

In this unit we learn regarding the history of mobile software development. We learn about the Wireless Application Protocol (WAP) and various mobile proprietary platforms.

## 1.5 Answers for Check Your Progress :

**1.** (C) **2.** (B) **3.** (A) **4.** (A) **5.** (C)

## 1.6 Glossary :

1. **Brick :** The Motorola DynaTAC 8000X was the first commercially available cell phone.
2. **WAP :** Wireless Application Protocol
3. **HTTP :** Hypertext Transfer Protocol
4. **WML :** Wireless Markup Language
5. **MMS :** Multimedia Messaging Service
6. **SMS :** Short Message Service
7. **BREW :** Binary Runtime Environment for Wireless
8. **EMS :** Enhanced Messaging Service

## 1.7 Assignment :

1. What is Android ?
2. Explain the history of mobile application development.
3. What are the various mobile proprietary platform available in market ?

## 1.8 Activities :

Think more about the history of mobile application development and collect the data for various mobile company introduce their first commercially available version.

## 1.9 Case Study :

Analysis the various mobile platform which support the WAP (Wireless Application Protocol) to run the website in mobile platforms.

## 1.10 Further Reading :

Lauren Darcey and Shane Conder, "Android Wireless Application Development", Pearson Education, 2nd ed. (2011).

Unit 02

# THE OPEN HANDSET ALLIANCE

## UNIT STRUCTURE

## 2.0 Learning Objectives :

**After learning this unit, you will be able to understand :**

- Formation of OHA (Open Handset Alliance)
- Android Platform architecture
- Various versions and Codename of Android

## 2.1 Introduction :

Enter search advertising giant Google. Now a household name, Google has shown an interest in spreading its vision, its brand, its search and ad–revenue–based platform, and its suite of tools to the wireless marketplace. The company's business model has been amazingly successful on the Internet and, technically speaking, wireless isn't that different.

### 2.1.1 Google Goes Wireless :

The company's initial forays into mobile were beset with all the problems you would expect. The freedoms Internet users enjoyed were not shared by mobile phone subscribers. Internet users can choose from the wide variety of computer brands, operating systems, Internet service providers, and web browser applications. Nearly all Google services are free and ad driven. Many applications in the Google Labs suite directly compete with the applications available on mobile phones. The applications range from simple calendars and calculators to navigation with Google Maps and the latest tailored news from News Alerts–not to mention corporate acquisitions such as Blogger and YouTube. When this approach didn't yield the intended results, Google decided to a different approach–to revamp the entire system upon which wireless application development was based, hoping to provide a more open environment for users and developers : the Internet model. The Internet model allows users to choose between freeware, shareware, and paid software. This enables free market competition among services.

## 2.2 Forming the Open Handset Alliance :

With its user–centric, democratic design philosophies, Google has led a movement to turn the existing closely guarded wireless market into one where phone users can move between carriers easily and have unfettered access to applications and services. With its vast resources, Google has taken a broad approach, examining the wireless infrastructure from the FCC wireless spectrum policies to the handset manufacturers' requirements, application developer needs, and mobile operator desires. Next, Google joined with other like–minded members in the wireless community and posed the following question : What would it take to build a better mobile phone ? The Open Handset Alliance (OHA) was formed in November 2007 to answer that very question. The OHA is a business alliance comprised of many of the largest and most successful mobile companies on the planet. Its members include chip makers, handset manufacturers, software developers, and service providers. The entire mobile supply chain is well represented. Andy Rubin has been credited as the father of the Android platform. His company, Android Inc., was acquired by Google in 2005.Working together, OHA members, including Google, began developing a nonproprietary open standard platform based upon technology developed at Android Inc. that would aim to alleviate the aforementioned problems hindering the mobile community. The result is the Android project. To this day, most Android platform development is completed by Rubin's team at Google, where he acts as VP of Engineering and manages the Android platform roadmap. Google's involvement in the Android project has been so extensive that the line between who takes responsibility for the Android platform (the OHA or Google) has blurred. Google hosts the Android open–source project and provides online Android documentation, tools, forums, and the Software Development Kit (SDK) for developers. All major Android news originates at Google. The company has also hosted a number of events at conferences and the Android Developer Challenge (ADC), a contest to encourage developers to write killer Android applications–for $10 million dollars in prizes to spur development on the platform. The winners and their apps are listed on the Android website.

## 2.3 Manufacturers : Designing the Android Handsets :

More than half the members of the OHA are handset manufacturers, such as Samsung, Motorola, HTC, and LG, and semiconductor companies, such as Intel, Texas Instruments, NVIDIA, and QUALCOMM. These companies are helping design the first generation of Android handsets. The first shipping Android handset–the T–Mobile G1–was developed by handset manufacturer HTC with service provided by T–Mobile. It was released in October 2008. Many other Android handsets were slated for 2009 and early 2010.The platform gained momentum relatively quickly. Each new Android device was more powerful and exciting than the last. Over the following 18 months, 60 different Android handsets (made by 21 different manufacturers) debuted across 59 carriers in 48 countries around the world. By June 2010, at an announcement of a new, highly anticipated Android handset, Google announced more than 160,000 Android devices were being activated each day (for a rate of nearly 60 million devices annually). The advantages of widespread manufacturer and carrier support appear to be really paying off at this point. The Android platform is now considered a success. It has shaken the mobile marketplace, gaining ground steadily against competitive platforms such as the Apple iPhone, RIM BlackBerry, and Windows Mobile. The latest numbers (as of Summer 2010) show BlackBerry in the lead with a declining 31% of the smartphone market. Trailing close behind is Apple's iPhone at 28%. Android, however, is trailing with 19%, though it's gaining ground rapidly and, according to some sources, is the fastest–selling smartphone platform. Microsoft Windows Mobile has been declining and now trails Android by several percentage points.

## 2.4 Mobile Operators : Delivering the Android Experience :

After you have the phones, you have to get them out to the users. Mobile operators from North, South, and Central America; Europe, Asia, India, Australia, Africa, and the Middle East have joined the OHA, ensuring a worldwide market for the Android movement. With almost half a billion subscribers alone, telephony giant China Mobile is a founding member of the alliance. Much of Android's success is also due to the fact that many Android handsets don't come with the traditional "smartphone price tag"–quite a few are offered free with activation by carriers. Competitors such as the Apple iPhone have no such offering as of yet. For the first time, the average Jane or Joe can afford a feature–full phone. I've lost count of the number of times I've had a waitress, hotel night manager, or grocery store checkout person tell me that they just got an Android phone and it has changed their life. This phenomenon has only added to the Android's rising underdog status. In the United States, the Android platform was given a healthy dose of help from carriers such as Verizon, who launched a $100 million dollar campaign for the first Droid handset. Many other Droid–style phones have followed from other carriers. Sprint recently launched the Evo 4G (America's first 4G phone) to much fanfare and record oneday sales (http://j.mp/cNhb4b)

## 2.5 Content Providers : Developing Android Applications :

When users have Android handsets, they need those killer apps, right ? Google has led the pack, developing Android applications, many of which, such as the email client and web browser, are core features of the platform. OHA members are also working on Android application integration. eBay, for example, is working on integration with its online auctions. The first ADC received 1,788 submissions, with the second ADC being voted upon by 26,000 Android users to pick a final 200 applications that would be judged professionally–all newly developed Android games, productivity helpers, and a slew of location based services (LBS) applications. We also saw humanitarian, social networking, and mash–up apps. Many of these applications have debuted with users through the Android Market–Google's software distribution mechanism for Android. For now, these challenges are over. The results, though, are still impressive. For those working on the Android platform from the beginning, handsets couldn't come fast enough. The T–Mobile G1 was the first commercial Android device on the market, but it had the air of a developer pre–release handset. Subsequent Android handsets have had much more impressive hardware, allowing developers to dive in and design awesome new applications As of October 2010, there are more than 80,000 applications available in the Android Market, which is growing rapidly. This takes into account only applications published through this one marketplace–not the many other applications sold individually or on other markets. This also does not take into account that, as of Android 2.2, Flash applications can run on Android handsets. This opens up even more application choices for Android users and more opportunities for Android developers. There are now more than 180,000 Android developers writing interesting and exciting applications. By the time you finish reading this book, you will be adding your expertise to this number.

## 2.6 Taking Advantage of All Android Has to Offer :

Android's open platform has been embraced by much of the mobile development community–extending far beyond the members of the OHA. As Android phones and applications have become more readily available, many other mobile operators and handset manufacturers have jumped at the chance to sell Android phones to their subscribers, especially given the cost benefits compared to proprietary platforms. The open standard of the Android platform has resulted in reduced operator costs in licensing and royalties, and we are now seeing a migration to open handsets from proprietary platforms such as RIM, Windows Mobile, and the Apple iPhone. The market has cracked wide open; new types of users are able to consider smartphones for the first time. Android is well suited to fill this demand.

## 2.7 Android Platform Differences :

Android is hailed as "the first complete, open, and free mobile platform" : n

- **Complete :** The designers took a comprehensive approach when they developed the Android platform. They began with a secure operating system and built a robust software framework on top that allows for rich application development opportunities.
- **Open :** The Android platform is provided through open–source licensing. Developers have unprecedented access to the handset features when developing applications.
- **Free :** Android applications are free to develop. There are no licensing or royalty fees to develop on the platform. No required membership fees. No required testing fees. No required signing or certification fees. Android applications can be distributed and commercialized in a variety of ways

### 2.7.1 Android : A Next–Generation Platform :

Although Android has many innovative features not available in existing mobile platforms, its designers also leveraged many tried–and–true approaches proven to work in the wireless world. It's true that many of these features appear in existing proprietary platforms, but Android combines them in a free and open fashion while simultaneously addressing many of the flaws on these competing platforms. The Android mascot is a little green robot, shown in Figure 2.1. This little guy (girl ?) is often used to depict Android–related materials. Android is the first in a new generation of mobile platforms, giving its platform developers a distinct edge on the competition. Android's designers examined the benefits and drawbacks of existing platforms and then incorporated their most successful features. At the same time, Android's designers avoided the mistakes others suffered in the past. Since the Android 1.0 SDK was released, Android platform development has continued at a fast and furious pace. For quite some time, there was a new Android SDK out every couple of months! In typical tech–sector jargon, each Android SDK has had a project name. In Android's case, the SDKs are named alphabetically after sweets (see Figure 2.2). The latest version of Android is codenamed Gingerbread.

**Figure 2.1 : The Android mascot and logo**

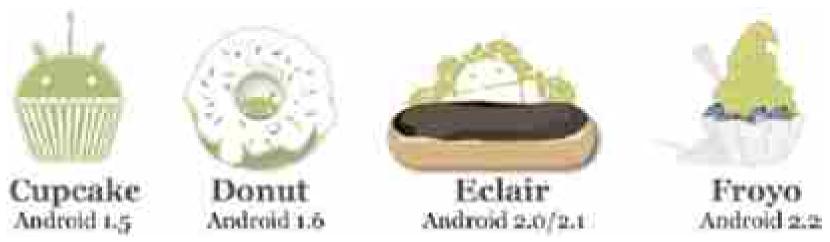


**Figure 2.2 : Some Android SDKs and their codenames**

## 2.8 Free and Open Source :

Android is an open–source platform. Neither developers nor handset manufacturers pay royalties or license fees to develop for the platform. The underlying operating system of Android is licensed under GNU General Public License Version 2 (GPLv2), a strong "copyleft" license where any third–party improvements must continue to fall under the open–source licensing agreement terms. The Android framework is distributed under the Apache Software License (ASL/Apache2), which allows for the distribution of both open– and closed–source derivations of the source code. Commercial developers (handset manufacturers especially) can choose to enhance the platform without having to provide their improvements to the open–source community. Instead, developers can profit from enhancements such as handset–specific improvements and redistribute their work under whatever licensing they want. Android application developers have the ability to distribute their applications under whatever licensing scheme they prefer. Developers can write open–source freeware or traditional licensed applications for profit and everything in between.

## 2.9 Familiar and Inexpensive Development Tools :

Unlike some proprietary platforms that require developer registration fees, vetting, and expensive compilers, there are no upfront costs to developing Android applications.

### 2.9.1 Freely Available Software Development Kit :

The Android SDK and tools are freely available. Developers can download the Android SDK from the Android website after agreeing to the terms of the Android Software Development Kit License Agreement.

### 2.9.2 Familiar Language, Familiar Development Environments :

Developers have several choices when it comes to integrated development environments (IDEs). Many developers choose the popular and freely available Eclipse IDE to design and develop Android applications. Eclipse is the most popular IDE for Android development, and there is an Android plug–in available for facilitating Android development. Android applications can be developed on the following operating systems :

- Windows XP (32–bit) or Vista (32–bit or 64–bit)
- Mac OS X 10.5.8 or later (x86 only)
- Linux (tested on Linux Ubuntu 8.04 LTS, Hardy Heron)

## 2.10 Reasonable Learning Curve for Developers :

Android applications are written in a well–respected programming language : Java.

The Android application framework includes traditional programming constructs, such as threads and processes and specially designed data structures to encapsulate objects commonly used in mobile applications. Developers can rely on familiar class libraries, such as `java.net` and `java.text.` Specialty libraries for tasks such as graphics and database management are implemented using well–defined open standards such as OpenGL Embedded Systems (OpenGL ES) or SQLite.

## 2.11 Enabling Development of Powerful Applications :

In the past, handset manufacturers often established special relationships with trusted third–party software developers (OEM/ODM relationships).This elite group of software developers wrote native applications, such as messaging and web browsers, which shipped on the handset as part of the phone's core feature set. To design these applications, the manufacturer would grant the developer privileged inside access and knowledge of a handset's internal software framework and firmware.

On the Android platform, there is no distinction between native and third–party applications, enabling healthy competition among application developers. All Android applications use the same libraries. Android applications have unprecedented access to the underlying hardware, allowing developers to write much more powerful applications. Applications can be extended or replaced altogether. For example, Android developers are now free to design email clients tailored to specific email servers, such as Microsoft Exchange or Lotus Notes.

## 2.12 Rich, Secure Application Integration :

Recall from the bat story I previously shared that I accessed a variety of phone applications in the course of a few moments : text messaging, phone dialer, camera, email, picture messaging, and the browser. Each was a separate application running on the phone– some built–in and some purchased. Each had its own unique user interface. None were truly integrated.

Not so with Android. One of the Android platform's most compelling and innovative features is well–designed application integration. Android provides all the tools necessary to build a better "bat trap," if you will, by allowing developers to write applications that seamlessly leverage core functionality such as web browsing, mapping, contact management, and messaging. Applications can also become content providers and share their data among each other in a secure fashion. Platforms such as Symbian have suffered from setbacks due to malware. Android's vigorous application security model helps protect the user and the system from malicious software.

## 2.13 No Costly Obstacles to Publication :

Android applications have none of the costly and time–intensive testing and certification programs required by other platforms such as BREW and Symbian.

## 2.14 A "Free Market" for Applications :

Android developers are free to choose any kind of revenue model they want. They can develop freeware, shareware, or trial–ware applications, ad–driven, and paid applications. Android was designed to fundamentally change the rules about what kind of wireless applications could be developed. In the past, developers faced many restrictions that had little to do with the application functionality or features :

- Store limitations on the number of competing applications of a given type
- Store limitations on pricing, revenue models, and royalties
- Operator unwillingness to provide applications for smaller demographics

With Android, developers can write and successfully publish any kind of application they want. Developers can tailor applications to small demographics, instead of just large–scale money–making ones often insisted upon by mobile operators. Vertical market applications can be deployed to specific, targeted users.

Because developers have a variety of application distribution mechanisms to choose from, they can pick the methods that work for them instead of being forced to play by others' rules. Android developers can distribute their applications to users in a variety of ways :

- Google developed the Android Market (see Figure 2.3), a generic Android application store with a revenue–sharing model.

**Figure 2.3 : The Android market.**

- Handango.com added Android applications to its existing catalogue using their billing models and revenue–sharing model.
- Developers can come up with their own delivery and payment mechanisms.

Mobile operators are still free to develop their own application stores and enforce their own rules, but it will no longer be the only opportunity developers have to distribute their applications.

## 2.15 A New and Growing Platform :

Android might be the next generation in mobile platforms, but the technology is still in its early stages. Early Android developers have had to deal with the typical roadblocks associated with a new platform : frequently revised SDKs, lack of good documentation, and market uncertainties.

On the other hand, developers diving into Android development now benefit from the first–to–market competitive advantages we've seen on other platforms such as BREW and Symbian. Early developers who give feedback are more likely to have an impact on the long–term design of the Android platform and what features will come in the next version of the SDK. Finally, the Android forum community is lively and friendly. Incentive programs, such as the ADC, have encouraged many new developers to dig into the platform.

Each new version of the Android SDK has provided a number of substantial improvements to the platform. In recent revisions, the Android platform has received some much needed UI "polish," both in terms of visual appeal and performance. Although most of these upgrades and improvements were welcome and necessary, new SDK versions often cause some upheaval within the Android developer community. A number of published applications have required retesting and resubmission to the Android Marketplace to conform to new SDK requirements, which are quickly rolled out to all Android phones in the field as a firmware upgrade, rendering older applications obsolete.

Some older Android handsets are not capable of running the latest versions of the platform. This means that Android developers often need to target several different SDK versions to reach all users. Luckily, the Android development tools make this easier than ever.

## 2.16 The Android Platform :

Android is an operating system and a software platform upon which applications are developed. A core set of applications for everyday tasks, such as web browsing and email, are included on Android handsets. As a product of the OHA's vision for a robust and open–source development environment for wireless, Android is an emerging mobile development platform. The platform was designed for the sole purpose of encouraging a free and open market that all mobile applications phone users might want to have and software developers might want to develop.

### 2.16.1 Android's Underlying Architecture :

The Android platform is designed to be more fault–tolerant than many of its predecessors. The handset runs a Linux operating system upon which Android applications are executed in a secure fashion. Each Android application runs in its own virtual machine (see Figure 2.4). Android applications are managed code; therefore, they are much less likely to cause the phone to crash, leading to fewer instances of device corruption (also called "bricking" the phone, or rendering it useless).

- **The Linux Operating System :** The Linux 2.6 kernel handles core system services and acts as a hardware abstraction layer (HAL) between the physical hardware of the handset and the Android software stack. Some of the core functions the kernel handles include
    - Enforcement of application permissions and security
    - Low–level memory management
    - Process management and threading
    - The network stack
    - Display, keypad input, camera, Wi–Fi, Flash memory, audio, and binder (IPC) driver access
- **Android Application Runtime Environment :** Each Android application runs in a separate process, with its own instance of the Dalvik virtual machine (VM). Based on the Java VM, the Dalvik design has been optimized for mobile devices. The Dalvik VM has a small memory footprint, and multiple instances of the Dalvik VM can run concurrently on the handset.

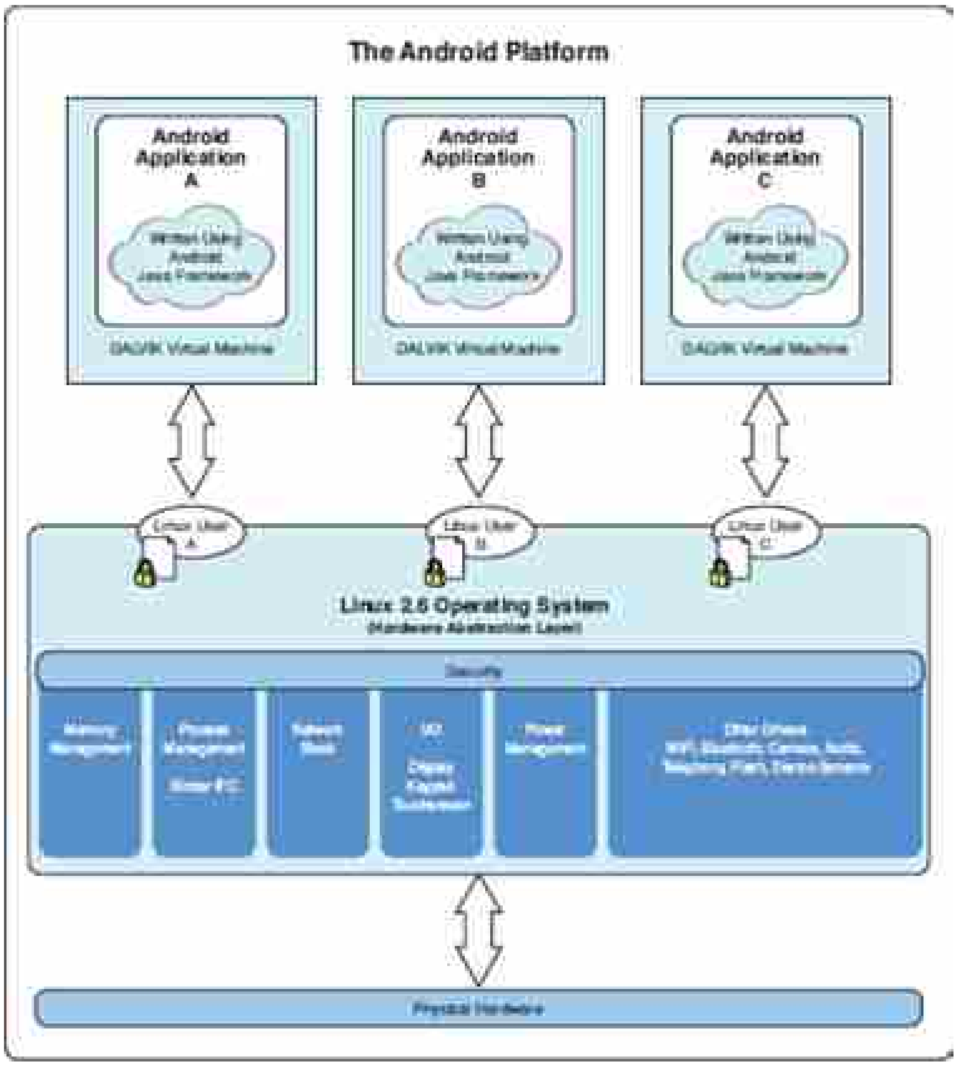


**Figure 2.4 : Diagram of the Android platform architecture**

**2.6.2 Security and Permissions :**

The integrity of the Android platform is maintained through a variety of security measures. These measures help ensure that the user's data is secure and that the device is not subjected to malware.

- **Applications as Operating System Users :** When an application is installed, the operating system creates a new user profile associated with the application. Each application runs as a different user, with its own private files on the file system, a user ID, and a secure operating environment. The application executes in its own process with its own instance of the Dalvik VM and under its own user ID on the operating system.
- **Explicitly Defined Application Permissions :** To access shared resources on the system, Android applications register for the specific privileges they require. Some of these privileges enable the application to use phone functionality to make calls, access the network, and control the camera and other hardware sensors. Applications also require permission to access shared data containing private and personal information, such as user preferences, user's location, and contact information.

  Applications might also enforce their own permissions by declaring them for other applications to use. The application can declare any number of different permission types, such as read–only or read–write permissions, for finer control over the application.

- **Limited Ad–Hoc Permissions :** Applications that act as content providers might want to provide some on–the–fly permissions to other applications for specific information, they want to share openly. This is done using ad–hoc granting and revoking of access to specific resources using Uniform Resource Identifiers (URIs).

  URIs index specific data assets on the system, such as images and text. Here is an example of a URI that provides the phone numbers of all contacts :

  ```
  content://contacts/phones
  ```

  To understand how this permission process works, let's look at an example.

  Let's say we have an application that keeps track of the user's public and private birthday wish lists. If this application wanted to share its data with other applications, it could grant URI permissions for the public wish list, allowing another application permission to access this list without explicitly having to ask for it.

- **Application Signing for Trust Relationships :** All Android applications packages are signed with a certificate, so users know that the application is authentic. The private key for the certificate is held by the developer. This helps establish a trust relationship between the developer and the user. It also enables the developer to control which applications can grant access to one another on the system. No certificate authority is necessary; self–signed certificates are acceptable.

- **Marketplace Developer Registration :** To publish applications on the popular Android Market, developers must create a developer account. The Android Market is managed closely and no malware is tolerated.

### 2.16.3 Developing Android Applications :

The Android SDK provides an extensive set of application programming interfaces (APIs) that is both modern and robust. Android handset core system services are exposed and accessible to all applications. When granted the appropriate permissions, Android applications can share data among one another and access shared resources on the system securely.

- **Android Programming Language Choices :** Android applications are written in Java (see Figure 2.5). For now, the Java language is the developer's only choice on the Android platform.

  There has been some speculation that other programming languages, such as C++, might be added in future versions of Android. If your application must rely on native code in another language such as C or C++, you might want to consider integrating it using the Android Native Development Kit (NDK).

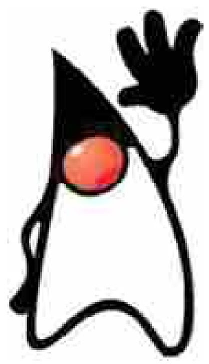


**Figure 2.5 : Duke, the Java mascot**

- **No Distinctions Made Between Native and Third–Party Applications :** Unlike other mobile development platforms, there is no distinction between native applications and developer–created applications on the Android platform. Provided the application is granted the appropriate permissions, all applications have the same access to core libraries and the underlying hardware interfaces.

  Android handsets ship with a set of native applications such as a web browser and contact manager. Third–party applications might integrate with these core applications, extend them to provide a rich user experience, or replace them entirely with alternative applications.

- **Commonly Used Packages :** With Android, mobile developers no longer have to reinvent the wheel. Instead, developers use familiar class libraries exposed through Android's Java packages to perform common tasks such as graphics, database access, network access, secure communications, and utilities (such as XML parsing).

  The Android packages include support for

  - Common user interface widgets (Buttons, Spin Controls, Text Input)
  - User interface layout
  - Secure networking and web browsing features (SSL, WebKit)
  - Structured storage and relational databases (SQLite)
  - Powerful 2D and 3D graphics (including SGL and OpenGL ES)
  - Audio and visual media formats (MPEG4, MP3, Still Images)
  - Access to optional hardware such as location–based services (LBS), Wi–Fi, Bluetooth, and hardware sensors

- **Android Application Framework :** The Android application framework provides everything necessary to implement your average application. The Android application lifecycle involves the following key components :

  - Activities are functions the application performs.
  - Groups of views define the application's layout.
  - Intents inform the system about an application's plans.
  - Services allow for background processing without user interaction.
  - Notifications alert the user when something interesting happens.

    Android applications can interact with the operating system and underlying hardware using a collection of managers. Each manager is responsible for keeping the state of some underlying system service. For example, there is a `LocationManager` that facilitates interaction with the location–based services available on the handset. The `ViewManager` and `WindowManager` manage user interface fundamentals.

    Applications can interact with one another by using or acting as a `ContentProvider`. Built–in applications such as the Contact manager are content providers, allowing third–party applications to access contact data and use it in an infinite number of ways. The sky is the limit.

## 2.17 Setting Up Your Android Development Environment :

Android developers write and test applications on their computers and then deploy those applications onto the actual device hardware for further testing. In this chapter, you become familiar with all the tools you need master in order to develop Android applications. You also explore the Android Software Development Kit (SDK) installation and all it has to offer.

### 2.17.1 Configuring Your Development Environment :

To write Android applications, you must configure your programming environment for Java development. The software is available online for download at no cost. Android applications can be developed on Windows, Macintosh, or Linux systems.

To develop Android applications, you need to have the following software installed on your computer :

- The Java Development Kit (JDK) Version 5 or 6, available for download at http://java.sun.com/javase/downloads/index.jsp.
- A compatible Java IDE such as Eclipse along with its JDT plug–in, available for download at http://www.eclipse.org/downloads/.
- The Android SDK, tools and documentation, available for download at http://developer.android.com/sdk/index.html.
- The Android Development Tools (ADT) plug–in for Eclipse, available for download through the Eclipse software update mechanism. For instructions on how to install this plug–in, see http://developer.android.com/sdk/eclipse–adt.html.Although this tool is optional for development, we highly recommend it and will use its features frequently throughout this book.

  A complete list of Android development system requirements is available at http://developer.android.com/sdk/requirements.html. Installation instructions are at http://developer.android.com/sdk/installing.html.

### 2.17.2 Configuring Your Operating System for Device Debugging :

To install and debug Android applications on Android devices, you need to configure your operating system to access the phone via the USB cable (see Figure 2.6). On some operating systems, such as Mac OS, this may just work. However, for Windows installations, you need to install the appropriate USB driver.You can download the Windows USB driver from the following website : http://developer.android.com/sdk/win–usb.html.

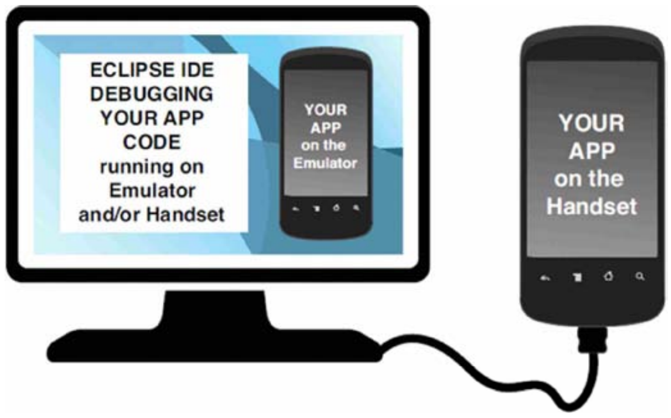

ECLIPSE IDE DEBUGGING YOUR APP CODE running on Emulator and/or Handset
YOUR APP on the Emulator
YOUR APP on the Handset

**Figure 2.6 : Android application debugging using the emulator and an Android handset.**

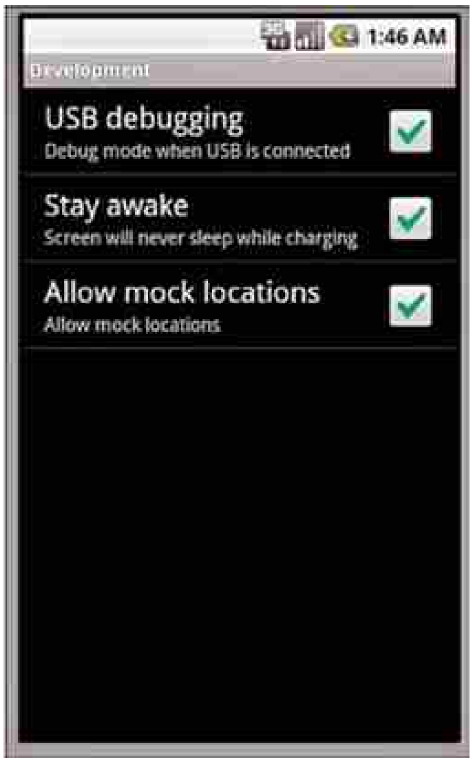
### 2.17.3 Configuring Your Android Hardware for Debugging :

Android devices have debugging disabled by default. Your Android device must be enabled for debugging via a USB connection in order to develop applications and run them on the device.

First, you need to enable your device to install Android applications other than those from the Android Market. This setting is reached by selecting Home, Menu, Settings, Applications. Here you should check (enable) the option called Unknown Sources.

More important development settings are available on the Android device by selecting Home, Menu, Settings, Applications, Development (see Figure 2.7). Here you should enable the following options :

- **USB Debugging :** This setting enables you to debug your applications via the USB connection.
- **Stay Awake :** This convenient setting keeps the phone from sleeping in the middle of your development work, as long as the device is plugged in.
- **Allow Mock Locations :** This setting enables you to send mock location information to the phone for development purposes and is very convenient for applications using location–based services (LBS).



USB debugging ✓
Debug mode when USB is connected

Stay awake ✓
Screen will never sleep while charging

Allow mock locations ✓
Allow mock locations

**Figure 2.7 : Android debug settings**

### 2.17.4 Upgrading the Android SDK :

The Android SDK is upgraded from time to time. You can easily upgrade the Android SDK and tools from within Eclipse using the Android SDK and AVD Manager, which is installed as part of the ADT plug–in for Eclipse.

Changes to the Android SDK might include addition, update, and removal of features; package name changes; and updated tools. With each new version of the SDK, Google provides the following useful documents :

- **An Overview of Changes :** A brief description of major changes to the SDK.
- **An API Diff Report :** A complete list of specific changes to the SDK.
- **Release Notes :** A list of known issues with the SDK.

You can find out more about adding and updating SDK components at http://developer.android.com/sdk/adding–components.html.

## 2.18 Getting to Know the Android Tools :

The Android SDK provides many tools to design, develop, debug, and deploy your Android applications.The Eclipse Plug–In incorporates many of these tools seamlessly into your development environment and provides various wizards for creating and debugging Android projects.

Settings for the ADT plug–in are found in Eclipse under Window, Preferences, Android. Here you can set the disk location where you installed the Android SDK and tools, as well as numerous other build and debugging settings.

The ADT plug–in adds a number of useful functions to the default Eclipse IDE.

Several new buttons are available on the toolbar, including buttons to

Launch the Android SDK and AVD Manager

- Create a new project using the Android Project Wizard
- Create a new test project using the Android Project Wizard
- Create a new Android XML resource file

These features are accessible through the Eclipse toolbar buttons shown in Figure 2.8.

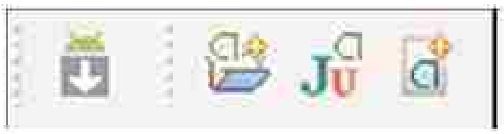
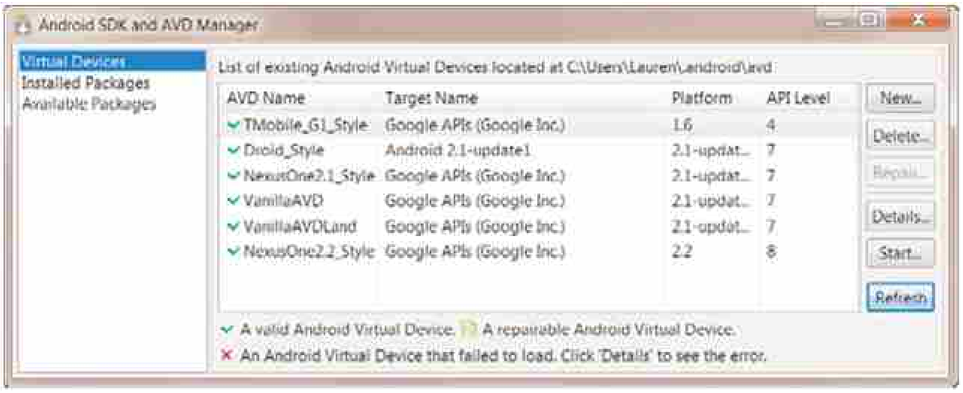


**Figure 2.8 : Android features added to the Eclipse toolbar**

There is also a special Eclipse perspective for debugging Android applications called DDMS (Dalvik Debug Monitor Server). You can switch to this perspective within Eclipse by choosing Window, Open Perspective, DDMS or by changing to the DDMS perspective in the top–right corner of the screen. We talk more about DDMS later in this chapter. After you have designed an Android application, you can also use the ADT plug–in for Eclipse to launch a wizard to package and sign your Android application for publication.

### 2.18.1 Android SDK and AVD Manager :

The Android SDK and AVD Manager, shown in Figure 2.9, is a tool integrated into Eclipse. This tool performs two major functions : management of multiple versions of the Android SDK on the development machine and management of the developer's Android Virtual Device (AVD) configurations.



Android SDK and AVD Manager

Virtual Devices
Installed Packages
Available Packages

List of existing Android Virtual Devices located at C:\Users\Lauren\.android\avd

| AVD Name | Target Name | Platform | API Level |
|---|---|---|---|
| ✓ TMobile_G1_Style | Google APIs (Google Inc.) | 1.6 | 4 |
| ✓ Droid_Style | Android 2.1-update1 | 2.1-updat... | 7 |
| ✓ NexusOne2.1_Style | Google APIs (Google Inc.) | 2.1-updat... | 7 |
| ✓ VanillaAVD | Google APIs (Google Inc.) | 2.1-updat... | 7 |
| ✓ VanillaAVDLand | Google APIs (Google Inc.) | 2.1-updat... | 7 |
| ✓ NexusOne2.2_Style | Google APIs (Google Inc.) | 2.2 | 8 |

New... Delete... Repair... Details... Start... Refresh

✓ A valid Android Virtual Device. A repairable Android Virtual Device.
✗ An Android Virtual Device that failed to load. Click 'Details' to see the error.

**Figure 2.9 : The Android SDK and AVD Manager**

Much like desktop computers, different Android devices run different versions of the Android operating system. Developers need to be able to target different Android SDK versions with their applications. Some applications target a specific Android SDK, whereas others try to provide simultaneous support for as many versions as possible.

The Android SDK and AVD Manager facilitate Android development across multiple platform versions simultaneously. When a new Android SDK is released, you can use this tool to download and update your tools while still maintaining backward compatibility and use older versions of the Android SDK.

The tool also manages the AVD configurations. To manage applications in the Android emulator, you must configure an AVD. This AVD profile describes what type of device you want the emulator to simulate, including which Android platform to support. You can specify different screen sizes and orientations, and you can specify whether the emulator has an SD card and, if so, what capacity.

**2.18.2 Android Emulator :**

The Android emulator, shown in Figure 2.10, is one of the most important tools provided with the Android SDK. You will use this tool frequently when designing and developing Android applications. The emulator runs on your computer and behaves much as a mobile device would. You can load Android applications into the emulator, test, and debug them.

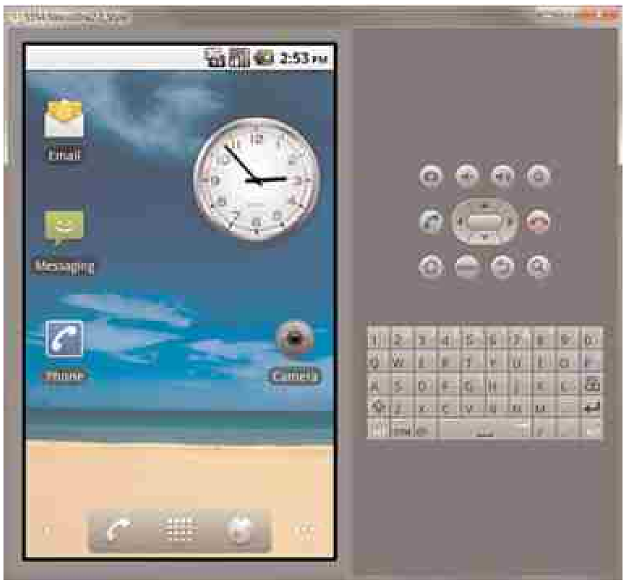


**Figure 2.10 : The Android emulator**

The emulator is a generic device and is not tied to any one specific phone configuration. You describe the hardware and software configuration details that the emulator is to simulate by providing an AVD configuration.

### 2.18.3 Dalvik Debug Monitor Server (DDMS) :

The Dalvik Debug Monitor Server (DDMS) is a command–line tool that has also been integrated into Eclipse as a perspective (see Figure 2.11). This tool provides you with direct access to the device–whether it's the emulator virtual device or the physical device. You use DDMS to view and manage processes and threads running on the device, view heap data, attach to processes to debug, and a variety of other tasks.

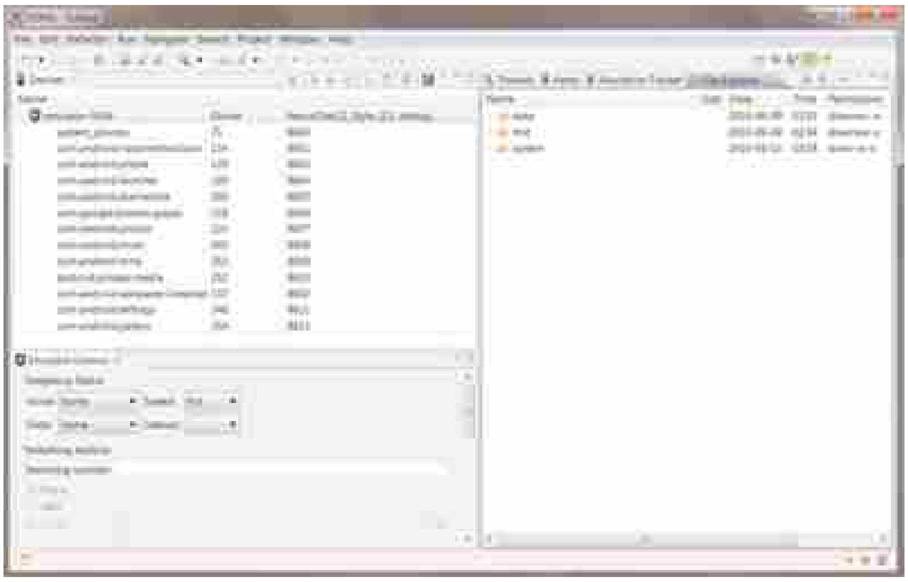


**Figure 2.11 : Using DDMS integrated into an Eclipse perspective**

### 2.18.4 Android Debug Bridge (ADB) :

The Android Debug Bridge (ADB) is a client–server tool used to enable developers to debug Android code on the emulator and the device using a standard Java IDE such as Eclipse. The DDMS and the Android Development Plug–In for Eclipse both use the ADB to facilitate interaction between the development environment and the device (or emulator).

Developers can also use ADB to interact with the device file system, install Android applications manually, and issue shell commands. For example, the sqlite3 shell commands enable you to access device database. The Application Exerciser Monkey commands generate random input and system events to stress test your application. One of the most important aspects of the ADB for the developer is its logging system (Logcat).

### 2.18.5 Android Hierarchy Viewer :

The Android Hierarchy Viewer (see Figure 2.12), a visual tool that illustrates layout component relationships, helps developers design and debug user interfaces. Developers can use this tool to inspect the View properties and develop pixel–perfect layouts. For more information about user interface design and the Hierarchy Viewer.

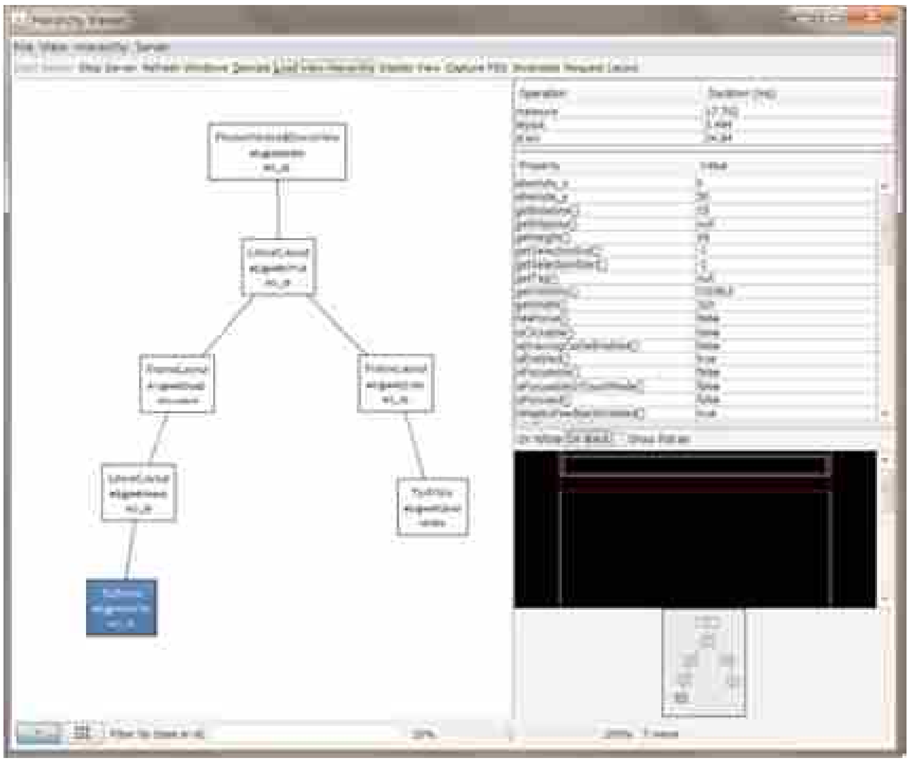


**Figure 2.12 : Screenshot of the Android Hierarchy Viewer in action**

❑ **Check Your Progress :**

1. OHA stands for ________

   (A) Open Handle Allaire (B) Open Handset Alliance

   (C) Opening Handling Allow (D) Open Hash Allowed

2. Android is ________

   (A) An Operating System (B) Phone

   (C) Device (D) Framework

3. Which virtual machine is used by the Android operating system ?

   (A) (B)

   (C) (D) Dalvik Virtual Machine

4. Android is based on which of the language ?

   (A) C++ (B) Java (C) Python (D) C

5. Which kernel is used in Android ?

   (A) Windows (B) MAC (C) Linux (D) Symbian

## 2.19 Let Us Sum Up :

In this unit we learn regarding the Open Handset Alliance. We learn about the Android Platform Architecture and various Codename or versions of the Android.

## 2.20 Answers for Check Your Progress :

**1.** (B) **2.** (A) **3.** (D) **4.** (B) **5.** (C)

## 2.21 Glossary :

**1. ADC :** Android Developer Challenge.

**2. LBS :** Location Based Service

3. **OHA :** Open Handset Alliance
4. **HAL :** Hardware Abstraction Layer
5. **SDK :** Software Development Kit
6. **AVD :** Android Virtual Device
7. **DDMS :** Dalvik Debug Monitor Server
8. **ADB :** Android Debug Bridge

## 2.22 Assignment :

1. What is OHA ?
2. Explain Android Platform Architecture in details .
3. What are the various versions of Android ? Explain it with proper codenames.

## 2.23 Activities :

1. Think more about the history of OHA and understand the mobile application development platform with its various versions.

## 2.24 Case Study :

Analysis the various mobile platform which support the various versions to run the website in mobile platforms.

## 2.25 Further Reading :

Lauren Darcey and Shane Conder, "Android Wireless Application Development", Pearson Education, 2nd ed. (2011).

Unit 03

# BUILDING A SAMPLE ANDROID APPLICATION

**UNIT STRUCTURE**

## 3.0 Learning Objectives :

- To develop the mobile application using Android
- To understand the basic of Android Application Development

## 3.1 Introduction :

You should now have a workable Android development environment set up on your computer. Hopefully, you also have an Android device as well. Now it's time for you to start writing some Android code. In this chapter, you learn how to add and create Android projects in Eclipse and verify that your Android development environment is set up correctly. You also write and debug your first Android application in the software emulator and on an Android handset.

## 3.2 Testing Your Development Environment :

The best way to make sure you configured your development environment correctly is to take an existing Android application and run it. You can do this easily by using one of the sample applications provided as part of the Android SDK in the /samples subdirectory. Within the Android SDK sample applications, you can find a classic game called Snake. To build and run the Snake application, you must create a new Android project in your Eclipse workspace, create an appropriate Android Virtual Device (AVD) profile, and configure a launch configuration for that project. After you have everything set up correctly, you can build the application and run it on the Android emulator or an Android device

**Adding the Application to a Project in Your Eclipse Workspace**

The first thing you need to do is add the Snake project to your Eclipse workspace.To do this, follow these steps :

1. Choose File, New, Project.
2. Choose Android,Android Project Wizard (see Figure 3.1).

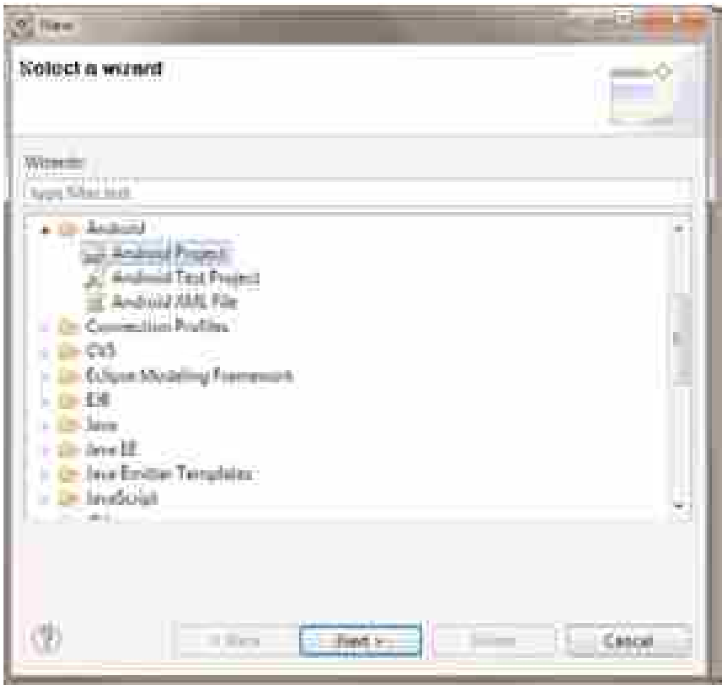


**Figure 3.1 : Creating a new Android project.**

3. Change the Contents to Create Project from Existing Source.
4. Browse to your Android samples directory.
5. Choose the Snake directory. All the project fields should be filled in for you from the Manifest file (see Figure 3.2). You might want to set the Build Target to whatever version of Android your test device is running.
6. Choose Finish. You now see the Snake project files in your workspace (see Figure 3.3).

**Creating an Android Virtual Device (AVD) for Your Project**

The next step is to create an AVD that describes what type of device you want to emulate when running the Snake application. This AVD profile describes what type of device you want the emulator to simulate, including which Android platform to support. You can specify different screen sizes and orientations, and you can specify whether the emulator has an SD card and, if it does, what capacity the card is.

Figure 3.2 : The project details.

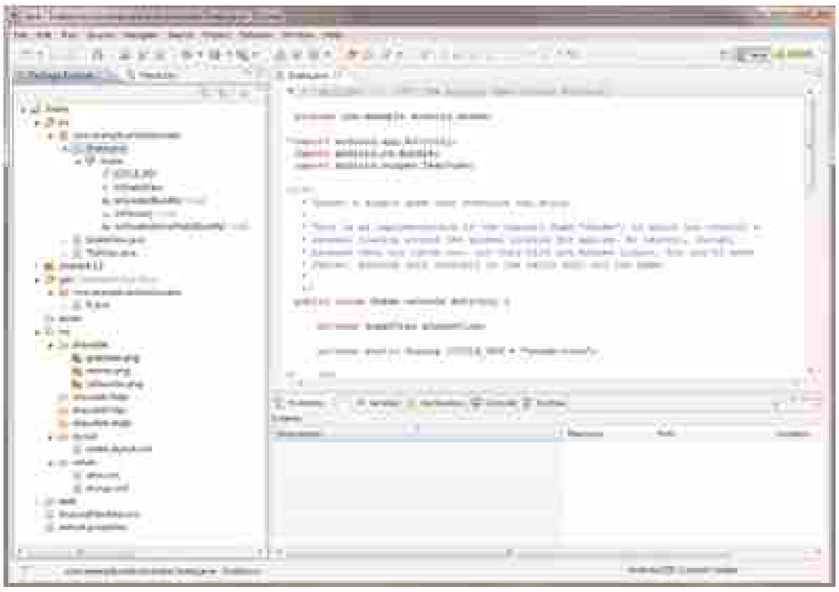


Figure 3.3 : The project files.

For the purposes of this example, an AVD for the default installation of Android 2.2 suffices. Here are the steps to create a basic AVD :

1. Launch the Android Virtual Device Manager from within Eclipse by clicking the little Android icon with the downward arrow on the toolbar. If you cannot find the icon, you can also launch the manager through the Window menu of Eclipse.
2. On the Virtual Devices menu, click the New button.
3. Choose a name for your AVD. Because we are going to take all the defaults, give this AVD a name of Android_Vanilla2.2.
4. Choose a build target. We want a basic Android 2.2 device, so choose Android 2.2 from the drop–down menu.

5. Choose an SD card capacity. This can be in kilobytes or megabytes and takes up space on your hard drive. Choose something reasonable, such as 1 gigabyte (1024M). If you're low on drive space or you know you won't need to test external storage to the SD card, you can use a much smaller value, such as 64 megabytes.
6. Choose a skin.This option controls the different resolutions of the emulator. In this case, use the WVGA800 screen.This skin most directly correlates to the popular Android handsets, such as the HTC Nexus One and the Evo 4G, both of which are currently sitting on my desk. Your project settings will look like Figure 3.4.
7. Click the Create AVD button, and wait for the operation to complete.
8. Click Finish. Because the AVD manager formats the memory allocated for SD card images, creating AVDs with SD cards could take a few moments.

**Creating a Launch Configuration for Your Project**

Next, you must create a launch configuration in Eclipse to configure under what circumstances the Snake application builds and launches. The launch configuration is where you configure the emulator options to use and the entry point for your application.

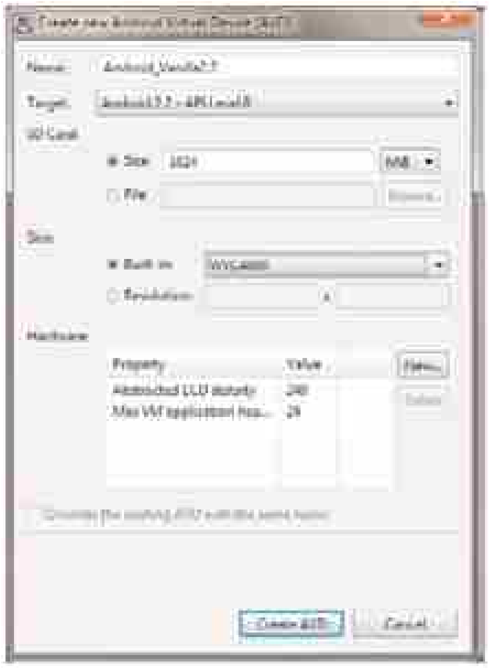


**Figure 3.4 : Creating a new AVD in Eclipse.**

You can create Run Configurations and Debug Configurations separately, each with different options. These configurations are created under the Run menu in Eclipse (Run, Run Configurations and Run, Debug Configurations).

Follow these steps to create a basic Run Configuration for the Snake application :

1. Choose Run, Run Configurations (or right–click the Project and choose Run As).
2. Double–click Android Application.
3. Name your Run Configuration SnakeRunConfiguration (see Figure 3.5).

4. Choose the project by clicking the Browse button and choosing the Snake project.
5. Switch to the Target tab and, from the preferred AVD list, choose the Android_Vanilla2.2 AVD created earlier, as shown in Figure 3.5.

You can set other options on the Target and Common tabs, but for now we are leaving the defaults as they are.

**Running the Snake Application in the Android Emulator**

Now you can run the Snake application using the following steps :

1. Choose the Run As icon drop–down menu on the toolbar (the green circle with the triangle).
2. Pull the drop–down menu and choose the SnakeRunConfiguration you created.
3. The Android emulator starts up; this might take a moment.

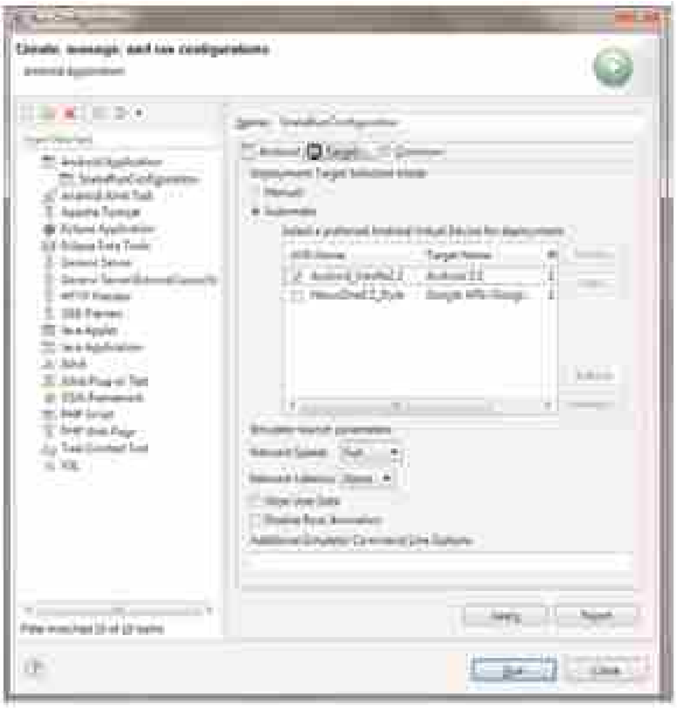


**Figure 3.5 : The Snake project launch configuration, Target tab with the AVD selected.**

4. If necessary, swipe the screen from left to right to unlock the emulator, as shown in Figure 3.6.
5. The Snake application now starts, as shown in Figure 3.7.

You can interact with the Snake application through the emulator and play the game. You can also launch the Snake application from the Application drawer at any time by clicking on its application icon.

## 3.3 Building Your First Android Application :

Now it's time to write your first Android application. You start with a simple Hello World project and build upon it to explore some of the features of Android.

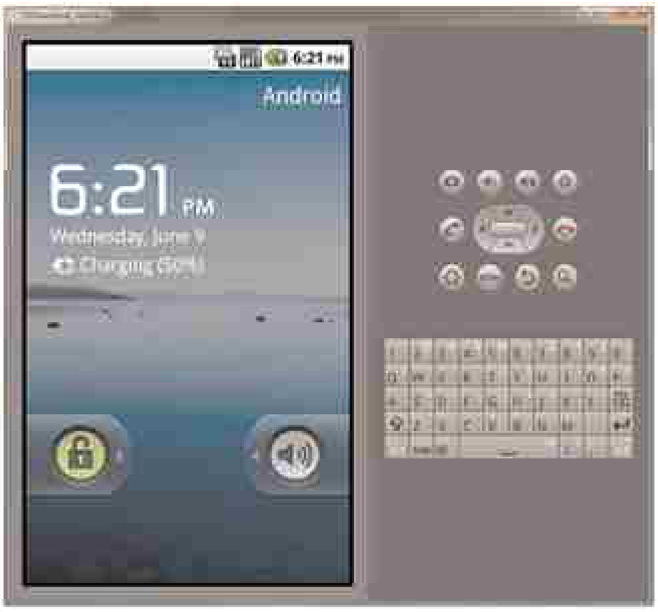


Figure 3.6 : The Android emulator launching (locked).

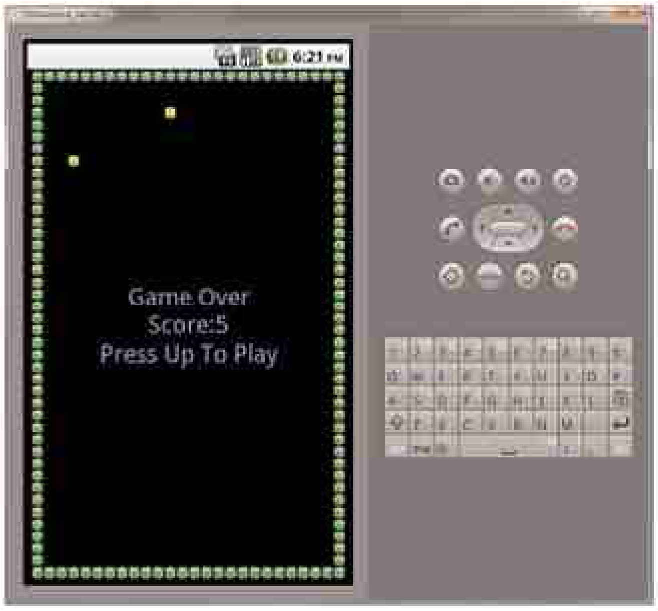


Figure 3.7 : The Snake game.

### 3.3.1 Creating and Configuring a New Android Project :

You can create a new Android project in much the same way as when you added the Snake application to your Eclipse workspace.

The first thing you need to do is create a new project in your Eclipse workspace. The Android Project Wizard creates all the required files for an Android application. Follow these steps within Eclipse to create a new project :

1. Choose File,New,Android Project, or choose the Android Project creator icon, which looks like a folder (with the letter a and a plus sign), on the Eclipse toolbar.
2. Choose a Project Name. In this case, name the project MyFirstAndroidApp.
3. Choose a Location for the project files. Because this is a new project, select the Create New Project in Workspace radio button. Check the Use Default Location checkbox or change the directory to wherever you want to store the source files.

4. Select a build target for your application. Choose a target that is compatible with the Android handsets you have in your possession. For this example, you might use the Android 2.2 target.
5. Choose an application name.The application name is the "friendly" name of the application and the name shown with the icon on the application launcher. In this case, the Application Name is My First Android App.
6. Choose a package name. Here you should follow standard package namespace conventions for Java. Because all our code examples in this book fall under the `com.androidbook.*` namespace, we will use the package name com.androidbook.myfirstandroidapp, but you are free to choose your own package name.
7. Check the Create Activity checkbox. This instructs the wizard to create a default launch activity for the application. Call this Activity class MyFirstAndroidAppActivity.

   Your project settings should look like Figure 3.8.
8. Finally, click the Finish button.

**3.3.2 Core Files and Directories of the Android Application :**

Every Android application has a set of core files that are created and are used to define the functionality of the application (see Table 3.1).The following files are created by default with a new Android application.

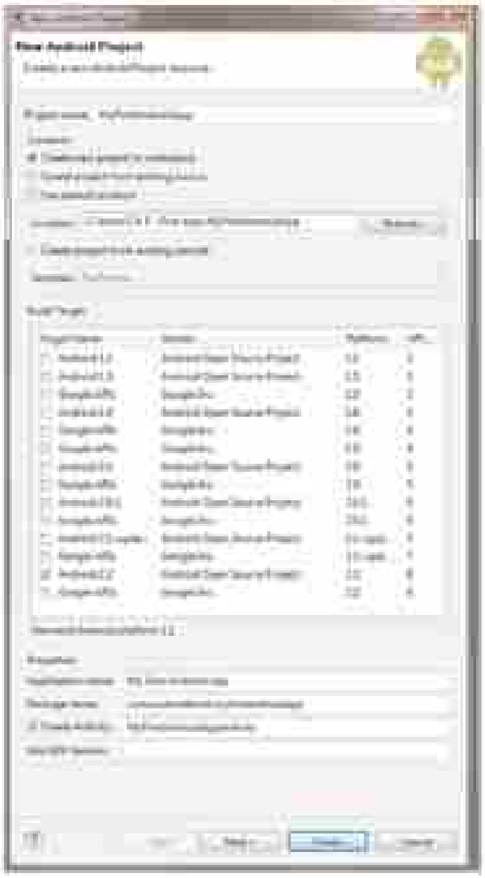


**Figure 3.8 : Configuring My First Android App using the Android Project Wizard.**

There are a number of other files saved on disk as part of the Eclipse project in the workspace. However, the files included in Table 3.1 are the important project files you will use on a regular basis.

### 3.3.3 Creating an AVD for Your Project :

The next step is to create an AVD that describes what type of device you want to emulate when running the application. For this example, we can use the AVD we created for the Snake application. An AVD describes a device, not an application. Therefore, you can use the same AVD for multiple applications. You can also create similar AVDs with the same configuration, but different data (such as different applications installed and different SD card contents).

**Table 3.1 Important Android Project Files and Directories**

| Android File | General Description |
|---|---|
| `AndroidManifest.xml` | Global application description file. It defines your application's capabilities and permissions and how it runs. |
| `default.properties` | Automatically created project file. It defines your application's build target and other build system options, as required. |
| `src Folder` | Required folder where all source code for the application resides. |
| `src/com.androidbook.myfirstandroidapp/MyFirstAndroidApp-Activity.java` | Core source file that defines the entry point of your Android application. |
| `gen Folder` | Required folder where auto-generated resource files for the application reside. |
| `gen/com.androidbook.myfirstandroidapp/R.java` | Application resource management source file generated for you; it should not be edited. |
| `res Folder` | Required folder where all application resources are managed. Application resources include animations, drawable image assets, layout files, XML files, data resources such as strings, and raw files. |
| `res/drawable-*/icon.png` | Resource folders that store different resolutions of the application icon. |
| `res/layout/main.xml` | Single screen layout file. |
| `res/values/strings.xml` | Application string resources. |
| `assets Folder` | Folder where all application assets are stored. Application assets are pieces of application data (files, directories) that you do not want managed as application resources. |

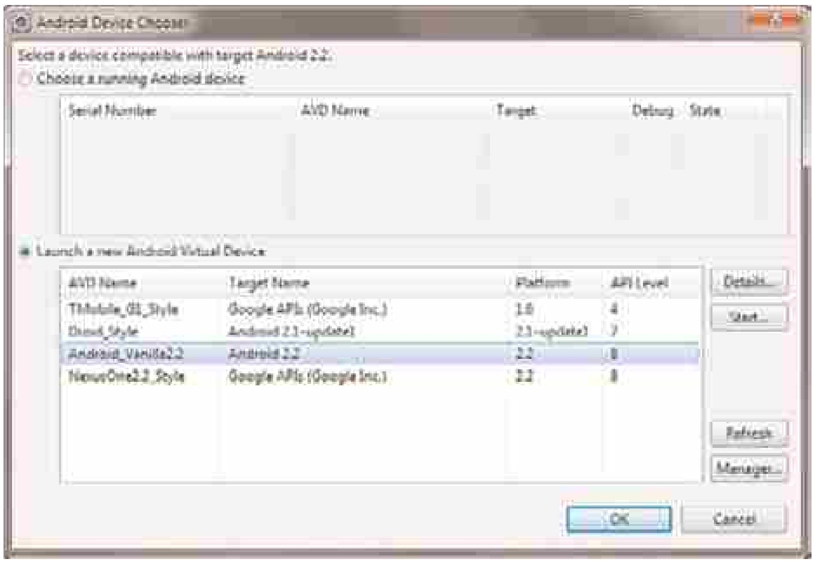
## 3.4 Running Your Android Application in the Emulator :

Now you can run the MyFirstAndroidApp application using the following steps :

1. Choose the Run As icon drop–down menu on the toolbar (the little green circle with the play button and a drop–down arrow) .
2. Pull the drop–down menu and choose the Run Configuration you created. (If you do not see it listed, choose the Run Configurations... item and

select the appropriate configuration. The Run Configuration shows up on this drop–down list the next time you run the configuration.)

3. Because you chose the Manual Target Selection mode, you are now prompted for your emulator instance. Change the selection to start a new emulator instance, and check the box next to the AVD you created, as shown in Figure 3.9.



Android Device Chooser

Select a device compatible with target Android 2.2.

○ Choose a running Android device

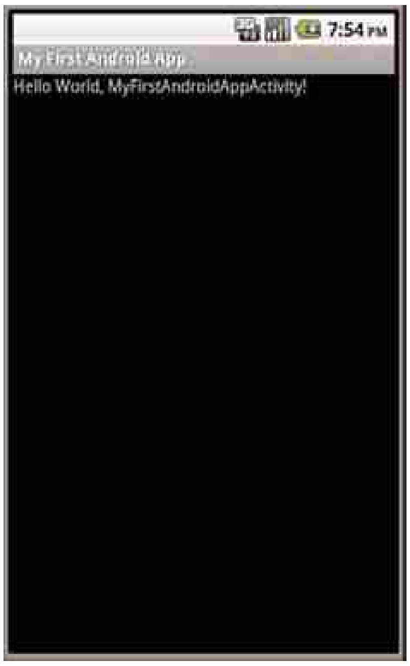
| Serial Number | AVD Name | Target | Debug | State |
|---|---|---|---|---|

● Launch a new Android Virtual Device

| AVD Name | Target Name | Platform | API Level |
|---|---|---|---|
| TMobile_G1_Style | Google APIs (Google Inc.) | 1.6 | 4 |
| Droid_Style | Android 2.1-update1 | 2.1-update1 | 7 |
| Android_Vanilla2.2 | Android 2.2 | 2.2 | 8 |
| NexusOne2.2_Style | Google APIs (Google Inc.) | 2.2 | 8 |

**Figure 3.9 : Manually choosing a target selection mode.**

4. The Android emulator starts up, which might take a moment.
5. Press the Menu button to unlock the emulator.
6. The application starts, as shown in Figure 3.10.
7. Click the Home button in the Emulator to end the application.
8. Pull up the Application Drawer to see installed applications. Your screen looks something like Figure 3.11.
9. Click on the My First Android Application icon to launch the application again.



My First Android App

Hello World, MyFirstAndroidAppActivity!

**Figure 3.10 : My First Android App running in the emulator.**

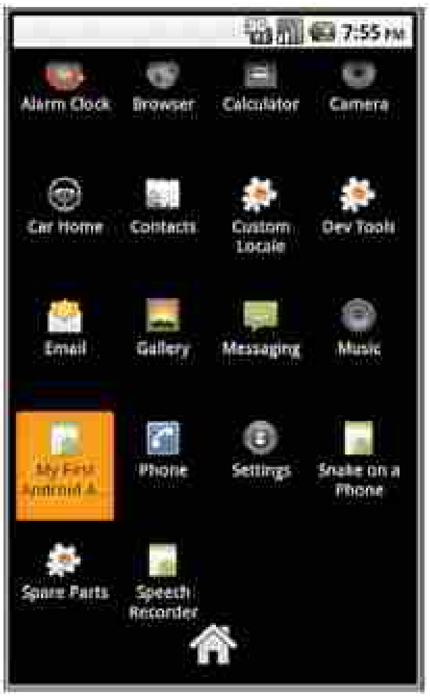


**Figure 3.11 : My First Android App application icon in the Drawer.**

### 3.4.1 Adding Logging Support to Your Android Application :

Before you start diving into the various features of the Android SDK, you should familiarize yourself with logging, a valuable resource for debugging and learning Android. Android logging features are in the Log class of the `android.util` package.

Some helpful methods in the `android.util.Log` class are shown in Table 3.2.

| Method | Purpose |
|---|---|
| `Log.e()` | Log errors |
| `Log.w()` | Log warnings |
| `Log.i()` | Log informational messages |
| `Log.d()` | Log Debug messages |
| `Log.v()` | Log Verbose mesages |

To add logging support to `MyFirstAndroidApp`, edit the file `MyFirstAndroidApp.java`.

First, you must add the appropriate import statement for the Log class :

```
import android.util.Log;
```

Next, within the `MyFirstAndroidApp` class, declare a constant string that you use to tag all logging messages from this class.You can use the LogCat utility within Eclipse to filter your logging messages to this debug tag :

```
private static final String DEBUG_TAG=
MyFirstAppLogging";
```

Now, within the onCreate() method, you can log something informational :

```
Log.i(DEBUG_TAG, "Info about MyFirstAndroidApp");
```

❑ **Check Your Progress :**

1. APK stands for ________

   (A) Android Pyramid Kit (B) Android Package Kit

   (C) Application Programming Kit (D) Android Prompted Kit

2. Where can we run the Android Program ?

   (A) Emulator or AVD (B) Command Prompt

   (C) Editor (D) Browser

3. Which file is contained in the src folder ?

   (A) XML file (B) Layout File

   (C) Java Source Code (D) None of the above

4. What is contained in manifest.xml ?

   (A) Android Code

   (B) Java Code

   (C) XML Tags

   (D) Permission that the application requires

5. Which of the method in android is used to log debug messages ?

   (A) Log.e() (B) Log.d() (C) Log.v() (D) Log.i()

## 3.5 Let Us Sum Up :

In this unit we leant how to develop sample and simple Android Application with its proper steps and its structure.

## 3.6 Answers for Check Your Progress :

**1.** (B) **2.** (A) **3.** (C) **4.** (D) **5.** (B)

## 3.7 Glossary :

**1.** **SDK :** Software Development Kit

**2.** **AVD :** Android Virtual Device

**3.** **DDMS :** Dalvik Debug Monitor Server

**4.** **ADB :** Android Debug Bridge

## 3.8 Assignment :

1. What is Android Application ?
2. Explain various steps to create Android Application.
3. Discuss how to compile and run android application in Emulator.

## 3.9 Activities :

1. Create Sample Android Application in Eclipse and Compile and Run in Android Emulator.

## 3.10 Case Study :

Think about the mobile application development in your surrounding area and try to analysis the content to be develop and use in the mobile application.

## 3.11 Further Reading :

Lauren Darcey and Shane Conder, "Android Wireless Application Development", Pearson Education, 2nd ed. (2011).

## BLOCK SUMMARY :

In this block we discussed regarding the various point covered in Units like Unit 1 History of Mobile Software Development, Unit 2 The Open Handset Alliance & Unit 3 Building a sample Android application. We have seen so far regarding the history of the Mobile Application Development and OHA as well as how to develop and run the simple mobile application in the android.

## BLOCK ASSIGNMENT :

1. Explain the history of mobile application development.
2. What are the various mobile proprietary platform available in market ?
3. What is OHA ?
4. What is Android ?
5. Explain Android Platform Architecture in details.
6. What are the various versions of Android ? Explain it with proper codenames.
7. What is Android Application ?
8. Explain various steps to create Android Application.
9. Discuss how to compile and run android application in Emulator.

❖ **Short Questions :**

1. What is Android ?
2. What is OHA ?
3. What is Android Application ?

❖ **Long Questions :**

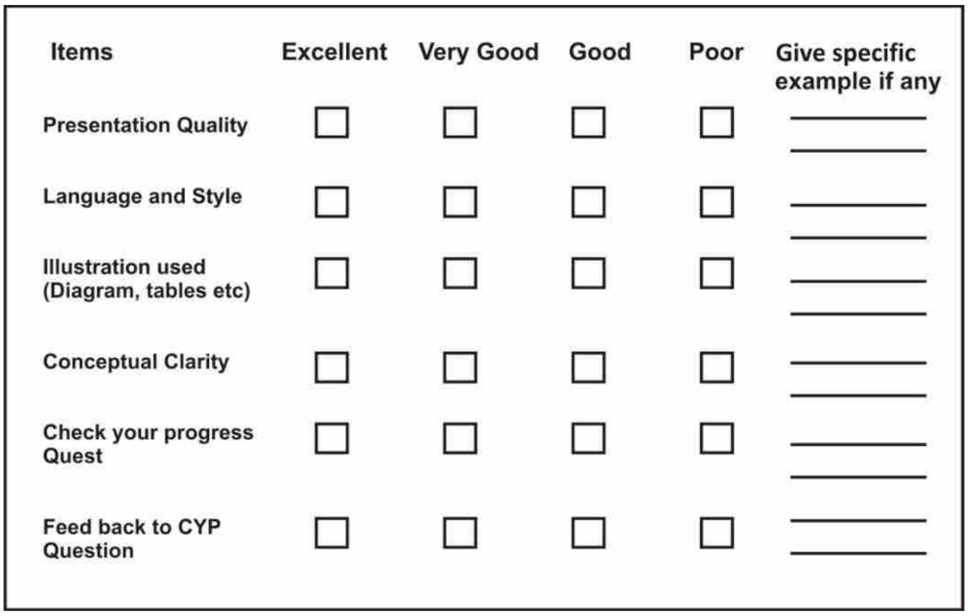
1. Explain the history of mobile application development.
2. What are the various mobile proprietary platform available in market ?
3. Explain Android Platform Architecture in details.
4. What are the various versions of Android ? Explain it with proper codenames.
5. Explain various steps to create Android Application.
6. Discuss how to compile and run android application in Emulator.

❖ **Enrolment No. :** ☐

1. How many hours did you need for studying the units ?

| Unit No. | 1 | 2 | 3 |
|---|---|---|---|
| No. of Hrs. | | | |

2. Please give your reactions to the following items based on your reading of the block :



| Items | Excellent | Very Good | Good | Poor | Give specific example if any |
|---|---|---|---|---|---|
| Presentation Quality | ☐ | ☐ | ☐ | ☐ | ____ |
| Language and Style | ☐ | ☐ | ☐ | ☐ | ____ |
| Illustration used (Diagram, tables etc) | ☐ | ☐ | ☐ | ☐ | ____ |
| Conceptual Clarity | ☐ | ☐ | ☐ | ☐ | ____ |
| Check your progress Quest | ☐ | ☐ | ☐ | ☐ | ____ |
| Feed back to CYP Question | ☐ | ☐ | ☐ | ☐ | ____ |

3. Any other Comments

.............................................................................................................

.............................................................................................................

.............................................................................................................

.............................................................................................................

.............................................................................................................

.............................................................................................................

.............................................................................................................

.............................................................................................................

**Dr. Babasaheb Ambedkar Open University Ahmedabad** **BCAR-503**

# *MOBILE APPLICATION DEVELOPMENT (USING ANDROID)*

**BLOCK 2 : ANDROID APPLICATION DESIGN ESSENTIALS**

# ANDROID APPLICATION DESIGN ESSENTIALS

**Block Introduction :**

Classical computer science classes often define a program in terms of functionality and data, and Android applications are no different. They perform tasks, display information to the screen, and act upon data from a variety of sources.

Developing Android applications for mobile devices with limited resources requires a thorough understanding of the application lifecycle. Android also uses its own terminology for these application building blocks-terms such as Context, Activity, and Intent. This chapter familiarizes you with the most important components of Android applications.

**Block Objectives :**

- To understand the anatomy of Android application
- To understand the Android Terminologies
- To understand the Application Context, Activities, Services & Intents
- To understand how to receive and broadcast the intents
- To use Android Manifest file and understand its common settings
- To use and set the various android permissions
- To understand how to Manage Application resources in a hierarchy & Working with different types of resources

## Block Structure :

Unit 04

# *ANATOMY OF AN ANDROID APPLICATIONS & ANDROID TERMINOLOGIES*

**UNIT STRUCTURE**

## 4.0 Learning Objectives :

- To learn how to implement the Context
- To learn how to create and use Activities
- To learn how to create and use Intents
- To learn how to create and use various Services

## 4.1 Introduction :

Classical computer science classes often define a program in terms of functionality and data, and Android applications are no different. They perform tasks, display information to the screen, and act upon data from a variety of sources.

Developing Android applications for mobile devices with limited resources requires a thorough understanding of the application lifecycle. Android also uses its own terminology for these application building blocks–terms such as Context, Activity, and Intent. This chapter familiarizes you with the most important components of Android applications.

## 4.2 Anatomy of Android Application & Terminologies :

### 4.2.1 Mastering Important Android Terminology :

This chapter introduces you to the terminology used in Android application development and provides you with a more thorough understanding of how Android applications function and interact with one another. Some of the important terms covered in this chapter are :

- **Context :** The context is the central command center for an Android application. All application–specific functionality can be accessed through the context.
- **Activity :** An Android application is a collection of tasks, each of which is called an Activity. Each Activity within an application has a unique task or purpose.
- **Intent :** The Android operating system uses an asynchronous messaging mechanism to match task requests with the appropriate Activity. Each request is packaged as an Intent. You can think of each such request as a message stating an intent to do something.
- **Service :** Tasks that do not require user interaction can be encapsulated in a service. A service is most useful when the operations are lengthy (offloading time–consuming processing) or need to be done regularly (such as checking a server for new mail).

### 4.2.2 Using the Application Context :

The application `Context` is the central location for all top–level application functionality. The Context class can be used to manage application–specific configuration details as well as application–wide operations and data. Use the application Context to access settings and resources shared across multiple `Activity` instances.

**Retrieving the Application Context**

You can retrieve the Context for the current process using the

getApplicationContext() method, like this:

```
Context context = getApplicationContext();
```

**Using the Application Context**

After you have retrieved a valid application Context, it can be used to access applicationwide features and services.

**Retrieving Application Resources**

You can retrieve application resources using the getResources() method of the application Context. The most straightforward way to retrieve a resource is by using its resource identifier, a unique number automatically generated within the R.java class. The following example retrieves a String instance from the application resources by its resource ID :

```
String greeting = getResources().getString
(R.string.hello);
```

We talk more about application resources in Chapter 6, "Managing Application Resources."

**Accessing Application Preferences**

You can retrieve shared application preferences using the getSharedPreferences() method of the application Context. The SharedPreferences class can be used to save simple application data, such as configuration settings.

We talk more about application preferences in Chapter 10,"Using Android Data and Storage APIs."

**Accessing Other Application Functionality Using Context**

The application Context provides access to a number of other top–level application features.

Here are a few more things you can do with the application Context :

- Launch Activity instances
- Retrieve assets packaged with the application
- Request a system service (for example, location service)
- Manage private application files, directories, and databases
- Inspect and enforce application permissions

The first item on this list–launching Activity instances–is perhaps the most common reason you use the application Context.

### 4.2.3 Performing Application Tasks with Activities :

The Android Activity class `(android.app.Activity)` is core to any Android application. Much of the time, you define and implement an `Activity` class for each screen in your application.

Activities are the most common of the four Android building blocks. An activity is usually a single screen in your application. Each activity is implemented as a single class that extends the Activity base class. Your class will display a user interface composed of Views and respond to events. Most applications consist of multiple screens. For example, a text messaging application might have one screen that shows a list of contacts to send messages to, a second screen to write the message to the chosen contact, and other screens to review old messages or change settings. Each of these screens would be implemented as an activity. Moving to another screen is accomplished by a starting a new activity. In some cases an Activity may return a value to the previous activity – for example an activity that lets the user pick a photo would return the chosen photo to the caller. When a new screen opens, the previous screen is paused and put onto a history stack. The user can navigate backward through previously opened screens in the history. Screens can also choose to be removed from the history stack when it would be inappropriate for them to remain. Android retains history stacks for each application launched from the home screen.

Activities are the most common of the four Android building blocks. An activity is usually a single screen in your application. Each activity is implemented as a single class that extends the Activity base class. Your class will display a user interface composed of Views and respond to events. Most applications consist of multiple screens. For example, a text messaging application might have one screen that shows a list of contacts to send messages to, a second screen to write the message to the chosen contact, and other screens to review old messages or change settings. Each of these screens would be

implemented as an activity. Moving to another screen is accomplished by a starting a new activity. In some cases, an Activity may return a value to the previous activity – for example an activity that lets the user pick a photo would return the chosen photo to the caller. When a new screen opens, the previous screen is paused and put onto a history stack. The user can navigate backward through previously opened screens in the history. Screens can also choose to be removed from the history stack when it would be inappropriate for them to remain. Android retains history stacks for each application launched from the home screen.

### 4.2.4 Managing Activity Transitions with Intents :

In the course of the lifetime of an Android application, the user might transition between a number of different `Activity` instances. At times, there might be multiple Activity instances on the activity stack. Developers need to pay attention to the lifecycle of each Activity during these transitions.

Some `Activity` instances–such as the application splash/startup screen–are shown and then permanently discarded when the Main menu screen Activity takes over. The user cannot return to the splash screen `Activity` without re–launching the application.

Other `Activity` transitions are temporary, such as a child Activity displaying a dialog box, and then returning to the original Activity (which was paused on the activity stack and now resumes). In this case, the parent Activity launches the child Activity and expects a result.

Android uses a special class called `Intent` to move from screen to screen. Intent describe what an application wants done. The two most important parts of the intent data structure are the action and the data to act upon. Typical values for action are MAIN (the front door of the application), VIEW, PICK, EDIT, etc. The data is expressed as a Uniform Resource Indicator (URI). For example, to view a website in the browser, you would create an Intent with the VIEW action and the data set to a Website–URI.

```
    new Intent(android.content.Intent.VIEW_ACTION,
              ContentURI.create("http://anddev.org"));
```

There is a related class called an `IntentFilter`. While an intent is effectively a request to do something, an intent filter is a description of what intents an activity (or intent receiver, see below) is capable of handling. An activity that is able to display contact information for a person would publish an IntentFilter that said that it knows how to handle the action VIEW when applied to data representing a person. Activities publish their IntentFilters in the `AndroidManifest.xml` file.

Navigating from screen to screen is accomplished by resolving intents. To navigate forward, an activity calls `startActivity(myIntent)`. The system then looks at the intent filters for all installed applications and picks the activity whose intent filters best matches `myIntent`. The new activity is informed of the intent, which causes it to be launched. The process of resolving intents happens at run time when startActivity is called, which offers two key benefits:

- Activities can reuse functionality from other components simply by making a request in the form of an Intent

- Activities can be replaced at any time by a new Activity with an equivalent IntentFilter

You can use an IntentReceiver when you want code in your application to execute in reaction to an external event, for example, when the phone rings, or when the data network is available, or when it's midnight. Intent receivers do not display a UI, although they may display Notifications to alert the user if something interesting has happened. Intent receivers are also registered in AndroidManifest.xml, but you can also register them from code using Context.registerReceiver(). Your application does not have to be running for its intent receivers to be called; the system will start your application, if necessary, when an intent receiver is triggered. Applications can also send their own intent broadcasts to others with Context.broadcastIntent().

### 4.2.5 Working with Services :

Trying to wrap your head around Activities, Intents, Intent Filters, and the lot when you start with Android development can be daunting. We have tried to distill everything you need to know to start writing Android applications with multiple Activity classes, but we'd be remiss if we didn't mention that there's a lot more here, much of which is discussed throughout the book using practical examples. However, we need to give you a "heads up" about some of these topics now because we talk about these concepts very soon when we cover configuring the Android Manifest file for your application in the next chapter.

One application component is the service. An Android Service is basically an Activity without a user interface. It can run as a background process or act much like a web service does, processing requests from third parties.

A `Service` is code that is long–lived and runs without a UI. A good example of this is a media player playing songs from a play list. In a media player application, there would probably be one or more activities that allow the user to choose songs and start playing them. However, the music playback itself should not be handled by an activity because the user will expect the music to keep playing even after navigating to a new screen. In this case, the media player activity could start a service using `Context.startService()` to run in the background to keep the music going. The system will then keep the music playback service running until it has finished. (You can learn more about the priority given to services in the system by reading Life Cycle of an Android Application.) Note that you can connect to a service (and start it if it's not already running) with the `Context.bindService()` method. When connected to a service, you can communicate with it through an interface exposed by the service. For the music service, this might allow you to pause, rewind, etc.

A `Service` is code that is long–lived and runs without a UI. A good example of this is a media player playing songs from a play list. In a media player application, there would probably be one or more activities that allow the user to choose songs and start playing them. However, the music playback itself should not be handled by an activity because the user will expect the music to keep playing even after navigating to a new screen. In this case, the media player activity could start a service using `Context.startService()` to run in the background to keep the music going. The system will then keep the music playback service running until it has finished. (You can learn more

about the priority given to services in the system by reading Life Cycle of an Android Application.) Note that you can connect to a service (and start it if it's not already running) with the `Context.bindService()` method. When connected to a service, you can communicate with it through an interface exposed by the service. For the music service, this might allow you to pause, rewind, etc.

## 4.3 Receiving and Broadcasting Intents :

Intents serve yet another purpose.You can broadcast an Intent object (via a call to broadcastIntent()) to the Android system, and any application interested can receive that broadcast (called a BroadcastReceiver).Your application might do both sending of and listening for Intent objects.These types of Intent objects are generally used to inform the greater system that something interesting has happened and use special Intent Action types.

For example, the Intent action ACTION_BATTERY_LOW broadcasts a warning when the battery is low. If your application is a battery–hogging Service of some kind, you might want to listen for this Broadcast and shut down your Service until the battery power is sufficient.You can register to listen for battery/ charge level changes by listening for the broadcast Intent object with the Intent action ACTION_BATTERY_CHANGED.There are also broadcast Intent objects for other interesting system events, such as SD card state changes, applications being installed or removed, and the wallpaper being changed.

Your application can also share information using the broadcast mechanism. For example, an email application might broadcast an Intent whenever a new email arrives so that other applications (such as spam or anti–virus apps) that might be interested in this type of event can react to it.

❑ **Check Your Progress :**

1. What is an Activity in Android ?

   (A) Activity Performs Data Analysis

   (B) Activity Performs Debugging

   (C) Activity Performs Compilation

   (D) Activity Performs Actions on the Screen

2. Which of the android component displays the part of an activity on screen ?

   (A) Segment (B) XML (C) Fragment (D) HTML

3. Which file specifies our screen's layout ?

   (A) XML (B) R.java (C) Source (D) Layout

4. In which state the activity is, if it is not in focus, but still visible on the screen ?

   (A) Pause State (B) Init State

   (C) Waiting State (D) Running State

5. Is it true that "There can be only one running activity at a given time" ?

   (A) True (B) False

## 4.4 Let Us Sum Up :

In this unit we learn regarding the anatomy of Android application, Android Terminologies, Application Context, Activities, Services & Intents, understand how to receive and broadcast the intents, use Android Manifest file and understand its common settings, use and set the various android permissions & understand how to Manage Application resources in a hierarchy & Working with different types of resources. We tried to strike a balance between providing a thorough reference without overwhelming you with details you won't need to know when developing the average Android application. Instead, we focused on the details you need to know to move forward developing Android applications and to understand every example provided within this book. Activity and View classes are the core building blocks of any Android application. Each Activity performs a specific task within the application, often with a single user interface screen consisting of View widgets. Each Activity is responsible for managing its own resources and data through a series of lifecycle callbacks. The transition from one Activity to the next is achieved through the Intent mechanism. An Intent object acts as an asynchronous message that the Android operating system processes and responds to by launching the appropriate Activity or Service. You can also use Intent objects to broadcast system–wide events to any interested BroadcastReceiver applications listening.

## 4.5 Answers for Check Your Progress :

**1.** (D) **2.** (C) **3.** (B) **4.** (A) **5.** (A)

## 4.6 Glossary :

1. **SDK :** Software Development Kit
2. **AVD :** Android Virtual Device
3. **DDMS :** Dalvik Debug Monitor Server
4. **ADB :** Android Debug Bridge

## 4.7 Assignment :

1. What is Activity ?
2. Define: Context
3. How to create Intent in Android ?
4. Define: Service

## 4.8 Activities :

1. Think about the various components available in the android system and link it with proper real–life problem and apply it.

## 4.9 Case Study :

1. Design and analysis of problem of application which can have various activities, intent & services.

## 4.10 Further Readings :

Android SDK Reference regarding the application Context class :

http://developer.android.com/reference/android/content/Context.html

Android SDK Reference regarding the Activity class :

http://developer.android.com/reference/android/app/Activity.html

Android Dev Guide:"Intents and Intent Filters" :

http://developer.android.com/guide/topics/intents/intents–filters.html

Android Programming with Tutorials from the anddev.org–Community written by Nicolas Gramlich

http://andbook.anddev.org

Unit 05

# APPLICATION CONTEXT, ACTIVITIES, SERVICES, INTENTS & RECEIVING AND BROADCASTING INTENTS

UNIT STRUCTURE

## 5.0 Learning Objectives :

- To learn how to implement the Context
- To learn how to create and use Activities
- To learn how to create and use Intents
- To learn how to create and use various Services

## 5.1 Introduction :

Android is an 'Open–source' operating system used to develop an application for mobile devices. It is a Linux based operating system. Initially, it was tenured by 'Open handset alliance' and in 2007 it was occupied by google. The source code published by Google is under the Apache License version 2.0. Android smartphones are used by millions of people worldwide. Under researches, researchers say that around 3 million people are using an android smartphone. The android application completely transforms the way to communicate and interact with each other even in the miles of distance.

## 5.2 Using the Application Context :

The application `Context` is the central location for all top–level application functionality. The Context class can be used to manage application–specific configuration details as well as application–wide operations and data. Use the application Context to access settings and resources shared across multiple `Activity` instances.

### 5.2.1 Retrieving the Application Context :

You can retrieve the Context for the current process using the `getApplicationContext()` method, like this:

```
Context context = getApplicationContext();
```

### 5.2.2 Using the Application Context :

After you have retrieved a valid application Context, it can be used to access applicationwide features and services.

- **Retrieving Application Resources :**

You can retrieve application resources using the `getResources()` method of the application `Context`. The most straightforward way to retrieve a resource is by using its resource identifier, a unique number automatically generated within the `R.java` class. The following example retrieves a String instance from the application resources by its resource ID :

```
String greeting =
getResources().getString(R.string.hello);
```

We talk more about application resources in next unit.

- **Accessing Application Preferences :**

You can retrieve shared application preferences using the `getSharedPreferences()` method of the application `Context`. The `SharedPreferences` class can be used to save simple application data, such as configuration settings.

- **Accessing Other Application Functionality Using Context :**

The application Context provides access to a number of other top–level application features.

Here are a few more things you can do with the application Context :

o Launch Activity instances

o Retrieve assets packaged with the application

o Request a system service (for example, location service) o Manage private application files, directories, and databases

o Inspect and enforce application permissions

The first item on this list–launching Activity instances is perhaps the most common reason you use the application Context.

### 5.2.3 Performing Application Tasks with Activities :

The Android Activity class `(android.app.Activity)` is core to any Android application. Much of the time, you define and implement an `Activity` class for each screen in your application.

For example, a simple game application might have the following five Activities, as shown in Figure 5.1 :

- **A Startup or Splash screen :** This activity serves as the primary entry point to the application. It displays the application name and version information and transitions to the Main menu after a short interval.
- **A Main Menu screen :** This activity acts as a switch to drive the user to the core Activities of the application. Here the users must choose what they want to do within the application.
- **A Game Play screen :** This activity is where the core game play occurs.
- **A High Scores screen :** This activity might display game scores or settings.
- **A Help/About screen :** This activity might display the information the user might need to play the game.

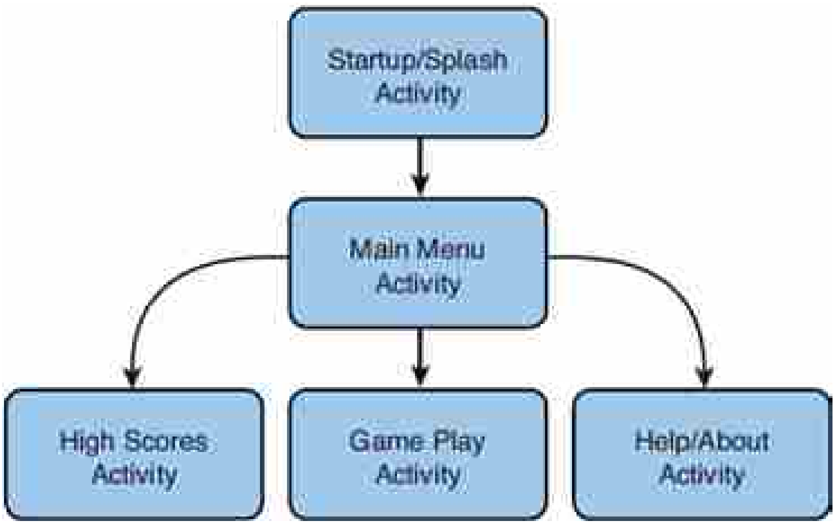


**Figure 5.1 : A simple game with five activities**

### 5.2.4 The Lifecycle of an Android Activity :

Android applications can be multi–process, and the Android operating system allows multiple applications to run concurrently, provided memory and processing power is available. Applications can have background processes, and applications can be interrupted and paused when events such as phone calls occur. There can be only one active application visible to the user at a time specifically, a single application Activity is in the foreground at any given time.

The Android operating system keeps track of all Activity objects running by placing them on an `Activity` stack (see Figure 5.2). When a new Activity starts, the Activity on the top of the stack (the current foreground Activity) pauses, and the new Activity pushes onto the top of the stack. When that Activity finishes, that Activity is removed from the activity stack, and the previous Activity in the stack resumes.

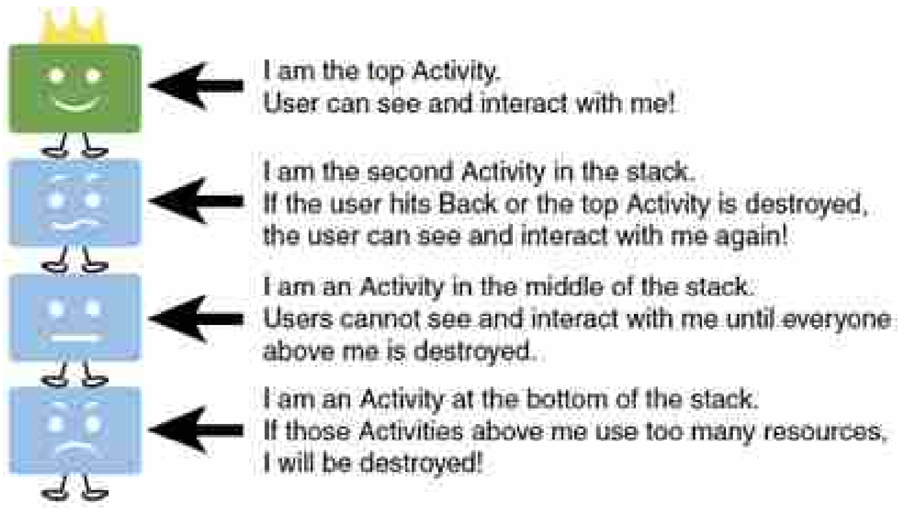

I am the top Activity.
User can see and interact with me!
I am the second Activity in the stack.
If the user hits Back or the top Activity is destroyed, the user can see and interact with me again!
I am an Activity in the middle of the stack.
Users cannot see and interact with me until everyone above me is destroyed.
I am an Activity at the bottom of the stack.
If those Activities above me use too many resources, I will be destroyed!

**Figure 5.2 : The Activity Stack**

Android applications are responsible for managing their state and their memory, resources, and data. They must pause and resume seamlessly. Understanding the different states within the Activity lifecycle is the first step to designing and developing robust Android applications.

### 5.2.5 Using `Activity` Callbacks to Manage Application State and Resources :

Different important state changes within the Activity lifecycle are punctuated by a series of important method callbacks. These callbacks are shown in Figure 5.3.

Here are the method stubs for the most important callbacks of the `Activity` class :

```
    public class MyActivity extends Activity {
        protected void onCreate(Bundle
savedInstanceState);
        protected void onStart();
        protected void onRestart();
        protected void onResume();
        protected void onPause();
        protected void onStop();
        protected void onDestroy();
        }
```

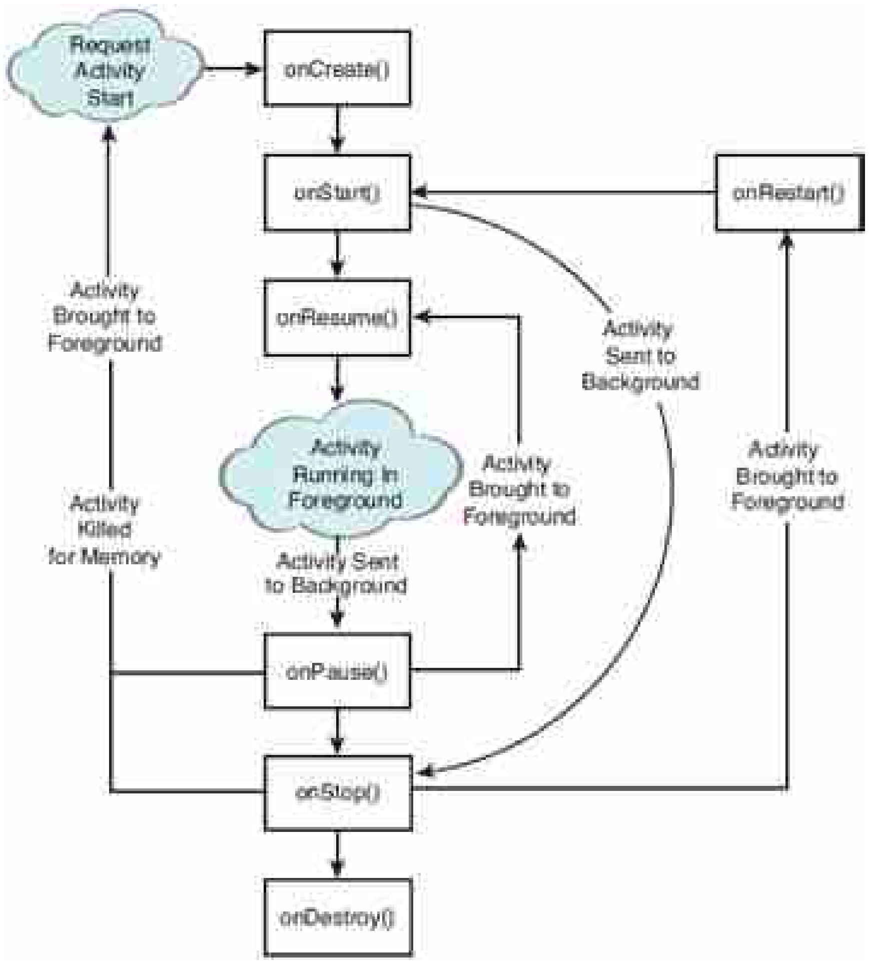

Request Activity Start
onCreate()
onStart()
onRestart()
Activity Brought to Foreground
onResume()
Activity Sent to Background
Activity Running In Foreground
Activity Brought to Foreground
Activity Brought to Foreground
Activity Killed for Memory
Activity Sent to Background
onPause()
onStop()
onDestroy()

**Figure 5.3 : The Life Cycle of an Android Activity**

Now let's look at each of these callback methods, when they are called, and what they

are used for.

- **Initializing Static Activity Data in `onCreate()`**
- **Initializing and Retrieving Activity Data in `onResume()`**
- **Stopping, Saving, and Releasing `Activity` Data in `onPause()`**
- **Saving `Activity` State into a `Bundle` with `onSaveInstanceState()`**
- **Destroy Static `Activity` Data in `onDestroy()`**

Check Your Progress

## 5.3 Managing Activity Transitions with Intents :

In the course of the lifetime of an Android application, the user might transition between a number of different Activity instances. At times, there might be multiple Activity instances on the activity stack. Developers need to pay attention to the lifecycle of each Activity during these transitions.

Some Activity instances such as the application splash/startup screen are shown and then permanently discarded when the Main menu screen Activity takes over. The user cannot return to the splash screen Activity without re–launching the application.

Other Activity transitions are temporary, such as a child Activity displaying a dialog box, and then returning to the original Activity (which was paused on the activity stack and now resumes). In this case, the parent Activity launches the child Activity and expects a result.

### 5.3.1 Transitioning Between Activities with Intents :

As previously mentioned, Android applications can have multiple entry points. There is no `main()` function, such as you find in iPhone development. Instead, a specific Activity can be designated as the main Activity to launch by default within the AndroidManifest.xml file; we talk more about this file in next chapter.

Other Activities might be designated to launch under specific circumstances. For example, a music application might designate a generic Activity to launch by default from the Application menu, but also define specific alternative entry point Activities for accessing specific music playlists by playlist ID or artists by name.

### 5.3.2 Launching a New Activity by Class Name :

You can start activities in several ways. The simplest method is to use the Application Context object to call the startActivity() method, which takes a single parameter, an Intent. An Intent (android.content.Intent) is an asynchronous message mechanism used by the Android operating system to match task requests with the appropriate Activity or Service (launching it, if necessary) and to dispatch broadcast Intents events to the system at large.

For now, though, we focus on Intents and how they are used with Activities. The following line of code calls the `startActivity()` method with an explicit Intent. This Intent requests the launch of the target Activity named `MyDrawActivity` by its class. This class is implemented elsewhere within the package.

```
startActivity(new Intent(getApplicationContext(),
MyDrawActivity.class));
```

This line of code might be sufficient for some applications, which simply transition from one Activity to the next. However, you can use the Intent mechanism in a much more robust manner. For example, you can use the Intent structure to pass data between Activities.

### 5.3.3 Creating Intents with Action and Data :

You've seen the simplest case to use an Intent to launch a class by name. Intents need not specify the component or class they want to launch explicitly. Instead, you can create an Intent Filter and register it within the Android Manifest file. The Android operating system attempts to resolve the Intent requirements and launch the appropriate Activity based on the filter criteria.

The guts of the Intent object are composed of two main parts: the action to be performed and the data to be acted upon.You can also specify action/ data pairs using Intent Action types and Uri objects.As you saw in Chapter 3,"Writing Your First Android Application," a Uri object represents a string that gives the location and name of an object.

Therefore, an Intent is basically saying "do this" (the action) to "that" (the Uri describing what resource to do the action to).

The most common action types are defined in the Intent class, including ACTION_MAIN (describes the main entry point of an Activity) and

ACTION_EDIT (used in conjunction with a Uri to the data edited).You also find Action types that generate integration points with Activities in other applications, such as the Browser or Phone Dialer.

### 5.3.4 Launching an Activity Belonging to Another Application :

Initially, your application might be starting only Activities defined within its own package. However, with the appropriate permissions, applications might also launch external Activities within other applications. For example, a Customer Relationship Management (CRM) application might launch the Contacts application to browse the Contact database, choose a specific contact, and return that Contact's unique identifier to the CRM application for use.

Here is an example of how to create a simple Intent with a predefined Action (ACTION_DIAL) to launch the Phone Dialer with a specific phone number to dial in the form of a simple Uri object:

```
Uri number = Uri.parse(tel:5555551212);
Intent dial = new Intent(Intent.ACTION_DIAL, number);
startActivity(dial);
```

You can find a list of commonly used Google application Intents athttp:/ /developer.android.com/guide/appendix/g–app–intents.html.Also available is the developer managed Registry of Intents protocols at OpenIntents, found at http:/ /www.openintents.org/en/intentstable, which has a growing list of Intents available from third–party applications and those within the Android SDK.

### 5.3.5 Passing Additional Information Using Intents :

You can also include additional data in an Intent. The Extras property of an Intent is stored in a Bundle object. The Intent class also has a number of helper methods for getting and setting name/value pairs for many common datatypes.

For example, the following Intent includes two extra pieces of information– a string value and a *boolean* :

```
Intent intent = new Intent(this, MyActivity.class);
intent.putExtra("SomeStringData","Foo");
intent.putExtra("SomeBooleanData",false);
```

### 5.3.6 Organizing Activities and Intents in Your Application Using Menus :

As previously mentioned, your application likely has a number of screens, each with its own Activity. There is a close relationship between menus, Activities, and Intents. You often see a menu used in two different ways with Activities and Intents:

- **Main Menu :** Acts as a switch in which each menu item launches a different Activity in your application. For instance, menu items for launching the Play Game *Activity*, the High Scores *Activity*, and the Help *Activity*.
- **Drill–Down :** Acts as a directory in which each menu item launches the same Activity, but each item passes in different data as part of the *Intent* (for example, a menu of all database records). Choosing a specific item might launch the Edit Record Activity, passing in that particular item's unique identifier.

## 5.4 Working with Services :

Trying to wrap your head around Activities, Intents, Intent Filters, and the lot when you start with Android development can be daunting. We have tried to distill everything you need to know to start writing Android applications with multiple *Activity* classes, but we'd be remiss if we didn't mention that there's a lot more here, much of which is discussed throughout the book using practical examples. However, we need to give you a "heads up" about some of these topics now because we talk about these concepts very soon when we cover configuring the Android Manifest file for your application in the next chapter.

One application component is the service. An Android Service is basically an Activity without a user interface. It can run as a background process or act much like a web service does, processing requests from third parties. You can use Intents and Activities to launch services using the *startService()* and *bindService()* methods. Any Services exposed by an Android application must be registered in the Android Manifest file.

You can use services for different purposes. Generally, you use a service when no input is required from the user. Here are some circumstances in which you might want to implement or use an Android service:

A weather, email, or social network app might implement a service to routinely check for updates. (**Note :** There are other implementations for polling, but this is a common use of services.)

- A photo or media app that keeps its data in sync online might implement a service to package and upload new content in the background when the device is idle.
- A video–editing app might offload heavy processing to a queue on its service in order to avoid affecting overall system performance for non–essential tasks.
- A news application might implement a service to "pre–load" content by downloading news stories in advance of when the user launches the application, to improve performance.

A good rule of thumb is that if the task requires the use of a worker thread and might affect application responsiveness and performance, consider implementing a service to handle the task outside the main application lifecycle.

## 5.5 Receiving and Broadcasting Intents :

Intents serve yet another purpose. You can broadcast an Intent object (via a call to *broadcastIntent()*) to the Android system, and any application interested can receive that broadcast (called a *BroadcastReceiver*).Your application might do both sending of and listening for Intent objects. These types of Intent objects are generally used to inform the greater system that something interesting has happened and use special Intent Action types.

For example, the Intent action **`ACTION_BATTERY_LOW`** broadcasts a warning when the battery is low. If your application is a battery–hogging Service of some kind, you might want to listen for this Broadcast and shut down your Service until the battery power is sufficient. You can register to listen for battery/charge level changes by listening for the broadcast Intent object with the Intent action **`ACTION_BATTERY_CHANGED`**. There are also broadcast

Intent objects for other interesting system events, such as SD card state changes, applications being installed or removed, and the wallpaper being changed.

Your application can also share information using the broadcast mechanism. For example, an email application might broadcast an Intent whenever a new email arrives so that other applications (such as spam or anti–virus apps) that might be interested in this type of event can react to it.

❑ **Check Your Progress :**

1. What does API stand for ?

   (A) Application Programming Interface

   (B) Application Program Intermediate

   (C) Applied Performance Indication

   (D) Application Paradigm Information

2. Which is the parent class of service ?

   (A) contextWriter (B) contextWrapper

   (C) contextRead (D) None of the Above

3. Which is used by services to clean up any services ?

   (A) onPause() method (B) onStart() method

   (C) onDestroy() method (D) onStop() method

4. R.java file is automatically generated file. (True/False)

   (A) True (B) False

5. What runs in Background and doesn't have any UI Components ?

   (A) Intent (B) Context (C) Components (D) Service

## 5.6 Let Us Sum Up :

In this unit we learn regarding how to implement the Context, how to create and use Activities, how to create and use Intents & how to create and use various Services.

## 5.7 Answers for Check Your Progress :

**1.** (A) **2.** (B) **3.** (C) **4.** (A) **5.** (D)

## 5.8 Glossary :

| | |
|---|---|
| **1. SDK :** | Software Development Kit |
| **2. AVD :** | Android Virtual Device |
| **3. DDMS :** | Dalvik Debug Monitor Server |
| **4. ADB :** | Android Debug Bridge |

## 5.9 Assignment :

1. What is Android? Explain various components of Android in details
2. Define Activity? Explain life cycle methods of android activity.
3. How to create intent in android ? Explain with suitable example.

## 5.10 Activities :

1. Search the real–life problem and analysis with help of android which can apply the mechanism of activity, intent & service.

## 5.11 Case Study :

Identify the real–life problem and solve the problems with help of android which can apply the mechanism of activity, intent & service.

## 5.12 Further Reading :

Lauren Darcey and Shane Conder, "Android Wireless Application Development", Pearson Education, 2nd ed. (2011).

Unit 06

# ANDROID MANIFEST FILE AND ITS COMMON SETTINGS & USING PERMISSION

## UNIT STRUCTURE

## 6.0 Learning Objectives :

- To learn the basic use of Android Manifest File in Android Application
- To learn how to set the Android Manifest File in Android Application
- To learn how to implement common setting in Android Manifest File
- To learn and use various permissions available in Android Application

## 6.1 Introduction :

Android projects use a special configuration file called the Android manifest file to determine application settings–settings such as the application name and version, as well as what permissions the application requires to run and what application components it is comprised of. In this chapter, you explore the Android manifest file in detail and learn how different applications use it to define and describe application behavior.

## 6.2 Configuring the Android Manifest File :

The Android application manifest file is a specially formatted XML file that must accompany each Android application. This file contains important information about the application's identity. Here you define the application's name and version information and what application components the application relies upon, what permissions the application requires to run, and other application configuration information.

The Android manifest file is named AndroidManifest.xml and must be included at the top level of any Android project. The information in this file is used by the Android system to

Install and upgrade the application package.

- Display the application details such as the application name, description, and icon to users.
- Specify application system requirements, including which Android SDKs are supported, what hardware configurations are required (for example, d–pad navigation), and which platform features the application relies upon (for example, uses multitouch capabilities).
- Launch application activities.
- Manage application permissions.
- Configure other advanced application configuration details, including acting as a service, broadcast receiver, or content provider.
- Enable application settings such as debugging and configuring instrumentation for application testing.

### 6.2.1 Editing the Android Manifest File :

The manifest resides at the top level of your Android project. You can edit the Android manifest file using the Eclipse Manifest File resource editor (a feature of the Android ADT plug–in for Eclipse) or by manually editing the XML.

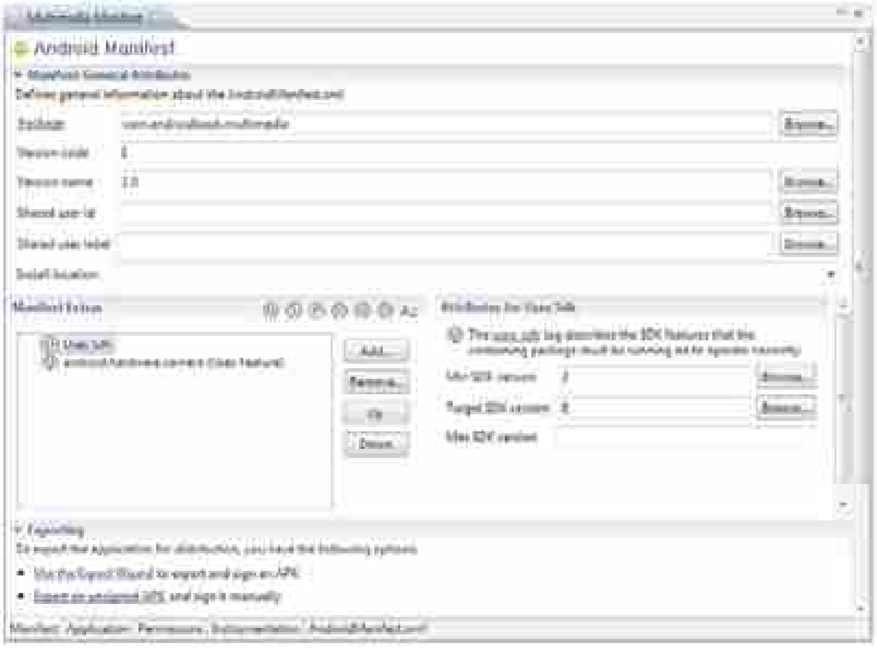
### 6.2.2 Editing the Manifest File Using Eclipse :

You can use the Eclipse Manifest File resource editor to edit the project manifest file. The Eclipse Manifest File resource editor organizes the manifest information into categories :

- The Manifest tab
- The Application tab
- The Permissions tab
- The Instrumentation tab
- The AndroidManifest.xml tab

Let's take a closer look at a sample Android manifest file. The figures and samples come from the Android application called Multimedia, which you build in upcoming chapter We chose this project because it illustrates a number of different characteristics of the Android manifest file, as opposed to the very simple default manifest file you configured for the MyFirstAndroidApp project.

o **Configuring Package–Wide Settings Using the Manifest Tab :** The Manifest tab (see Figure 6.1) contains package–wide settings, including the package name, version information, and supported Android SDK information. You can also set any hardware or feature requirements here.



Android Manifest

Manifest General Attributes

Defines general information about the AndroidManifest.xml

Package: com.androidbook.multimedia

Version code: 1

Version name: 1.0

Shared user id:

Shared user label:

Install location:

Manifest Extras

Uses Sdk

android.hardware.camera (Uses Feature)

Attributes for Uses Sdk

The uses-sdk tag describes the SDK features that the containing package must be running on to operate correctly.

Min SDK version: 7

Target SDK version: 8

Max SDK version:

Exporting

To export the application for distribution, you have the following options:

- Use the Export Wizard to export and sign an APK
- Export an unsigned APK and sign it manually

**Figure 6.1 The Manifest tab of the Eclipse Manifest File resource editor.**

o **Managing Application and Activity Settings Using the Application Tab :** The Application tab (see Figure 6.2) contains application–wide settings, including the application label and icon, as well as information about the application components such as activities, intent filters, and other application components, including configuration for services, intent filters, and content providers.

o **Enforcing Application Permissions Using the Permissions Tab :** The Permissions tab (see Figure 6.3) contains any permission rules required by your application. This tab can also be used to enforce custom permissions created for the application.

o **Managing Test Instrumentation Using the Instrumentation Tab :** The Instrumentation tab allows the developer to declare any instrumentation classes for monitoring the application. We talk more about instrumentation and testing in upcoming Chapter.

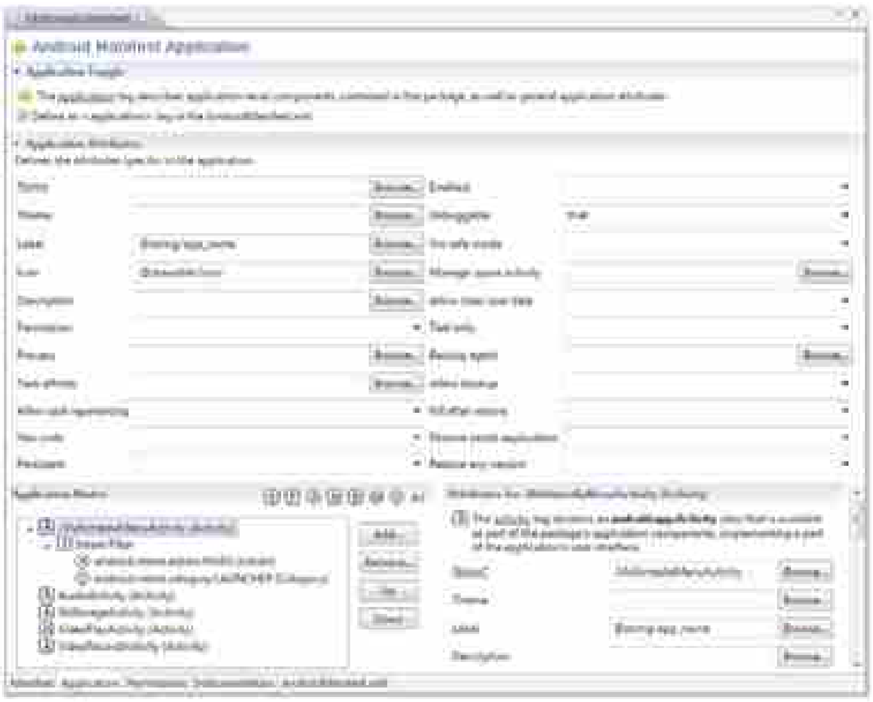


Figure 6.2 : The Application tab of the Eclipse Manifest File resource editor.

Figure 6.3 : The Permissions tab of the Eclipse Manifest File resource editor.

### 6.2.3 Editing the Manifest File Manually :

The Android manifest file is a specially formatted XML file. You can edit the XML manually by clicking on the AndroidManifest.xml tab. Android manifest files generally include a single <manifest> tag with a single <application> tag. The following is a sample AndroidManifest.xml file for an application called Multimedia :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidbook.multimedia"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name"
        android:debuggable="true">
        <activity android:name=".MultimediaMenuActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action
                    android:name="android.intent.action.MAIN" />
                <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="AudioActivity"></activity>
        <activity android:name="StillImageActivity"></activity>
        <activity android:name="VideoPlayActivity"></activity>
        <activity android:name="VideoRecordActivity"></activity>
    </application>
    <uses-permission
        android:name="android.permission.WRITE_SETTINGS" />
    <uses-permission
        android:name="android.permission.RECORD_AUDIO" />
    <uses-permission
        android:name="android.permission.SET_WALLPAPER" />
    <uses-permission
        android:name="android.permission.CAMERA"></uses-permission>
    <uses-sdk
        android:minSdkVersion="3"
        android:targetSdkVersion="8">
    </uses-sdk>
    <uses-feature
         android:name="android.hardware.camera" />
</manifest>
```

Here's a summary of what this file tells us about the Multimedia application :

- The application uses the package name com.androidbook.multimedia.
- The application version name is 1.0.
- The application version code is 1.
- The application name and label are stored in the resource string called @string/app_name within the /res/values/strings.xml resource file.
- The application is debuggable on an Android device.
- The application icon is the graphic file called icon (could be a PNG, JPG, or GIF) stored within the /res/drawable directory (there are actually multiple versions for different pixel densities).
- The application has five activities (MultimediaMenuActivity, AudioActivity, StillImageActivity, VideoPlayActivity, and VideoRecordActivity).
- MultimediaMenuActivity is the primary entry point for the application.This is the activity that starts when the application icon is pressed in the application drawer.
- The application requires the following permissions to run: the ability to record audio, the ability to set the wallpaper on the device, the ability to access the built–in camera, and the ability to write settings.
- The application works from any API level from 3 to 8; in other words,Android SDK 1.5 is the lowest supported, and the application was written to target Android 2.2.
- Finally, the application requires a camera to work properly

Now let's talk about some of these important configurations in detail.

## 6.3 Managing Your Application's Identity :

Your application's Android manifest file defines the application properties. The package name must be defined within the Android manifest file within the <manifest> tag using the package attribute :

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidbook.multimedia"
    android:versionCode="1"
    android:versionName="1.0">
```

### 6.3.1 Versioning Your Application :

Versioning your application appropriately is vital to maintaining your application in the field. Intelligent versioning can help reduce confusion and make product support and upgrades simpler. There are two different version attributes defined within the <*manifest*> tag: the version name and the version code.

The version name *(android:versionName)* is a user–friendly, developer–defined version attribute. This information is displayed to users when they manage applications on their devices and when they download the application from marketplaces. Developers use this version information to keep track of their application versions in the field. We discuss appropriate application versioning for mobile applications in detail in Chapter

### 6.3.2 Setting the Application Name and Icon :

Overall application settings are configured with the <application> tag of the Android manifest file. Here you set information such as the application icon *(android:icon)* and friendly name *(android:label)*.These settings are attributes of the <*application*> tag.

For example, here we set the application icon to a drawable resource provided with the application package and the application label to a string resource:

*<application android:icon="@drawable/icon"*

*android:label="@string/app_name">*

You can also set optional application settings as attributes in the <application> tag, such as the application description *(android:description)* and the setting to enable the application for debugging on the device *(android:debuggable="true")*.

## 6.4 Enforcing Application System Requirements :

In addition to configuring your application's identity, the Android manifest file is also used to specify any system requirements necessary for the application to run properly. For example, an augmented reality application might require that the handset have GPS, a compass, and a camera. Similarly, an application that relies upon the Bluetooth APIs available within the Android SDK requires a handset with an SDK version of API Level 5 or higher (Android 2.0). These types of system requirements can be defined and enforced in the Android manifest file. Then, when an application is listed on the Android Market, applications can be filtered by these types of information; the Android platform also checks these requirements when installing the application package on the system and errors out if necessary.

Some of the application system requirements that developers can configure through the Android manifest file include

- The Android SDK versions supported by the application
- The Android platform features used by the application
- The Android hardware configurations required by the application
- The screen sizes and pixel densities supported by the application
- Any external libraries that the application links to

**6.4.1 Targeting Specific SDK Versions :**

Android devices run different versions of the Android platform. Often, you see old, less powerful, or even less expensive devices running older versions of the Android platform, whereas newer, more powerful devices that show up on the market often run the latest Android software.

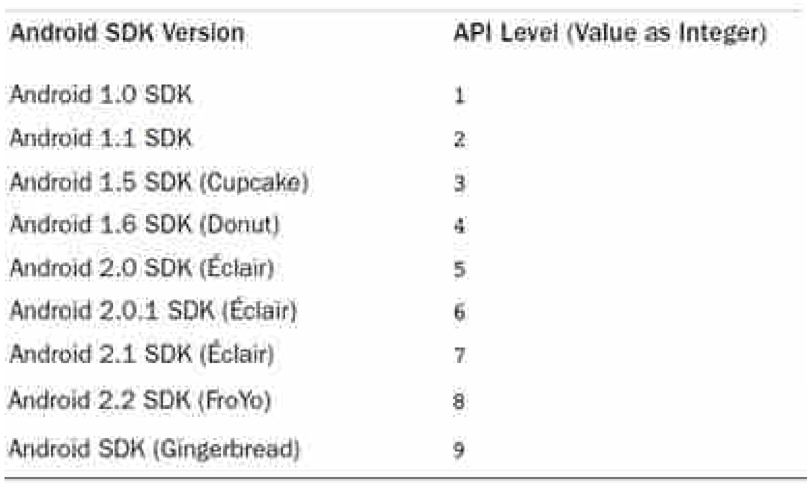
There are now dozens of different Android devices in users' hands. Developers must decide who their target audience is for a given application. Are they trying to support the largest population of users and therefore want to support as many different versions of the platform as possible ? Or are they developing a bleeding–edge game that requires the latest device hardware?

Developers can specify which versions of the Android platform an application supports within its Android manifest file using the <uses–sdk> tag.This tag has three important attributes:

- The **minSdkVersion** attribute : This attribute specifies the lowest API level that the application supports.
- The **targetSdkVersion** attribute : This attribute specifies the optimum API level that the application supports.
- The **maxSdkVersion** attribute : This attribute specifies the highest API level that the application supports.

Each attribute of the **<uses–sdk>** tag is an integer that represents the API level associated with a given Android SDK. This value does not directly correspond to the SDK version. Instead, it is the revision of the API level associated with that SDK. The API level is set by the developers of the Android SDK. You need to check the SDK documentation to determine the API level value for each version.

Table 6.4 shows the Android SDK versions available for shipping applications.



| Android SDK Version | API Level (Value as Integer) |
|---|---|
| Android 1.0 SDK | 1 |
| Android 1.1 SDK | 2 |
| Android 1.5 SDK (Cupcake) | 3 |
| Android 1.6 SDK (Donut) | 4 |
| Android 2.0 SDK (Éclair) | 5 |
| Android 2.0.1 SDK (Éclair) | 6 |
| Android 2.1 SDK (Éclair) | 7 |
| Android 2.2 SDK (FroYo) | 8 |
| Android SDK (Gingerbread) | 9 |

### 6.4.2 Specifying the Minimum SDK Version :

You should always specify the *minSdkVersion* attribute for your application. This value represents the lowest Android SDK version your application supports.

For example, if your application requires APIs introduced in Android SDK 1.6, you would check that SDK's documentation and find that this release is defined as API Level 4. Therefore, add the following to your Android Manifest file within the <manifest> tag block :

*<uses–sdk android:minSdkVersion="4" />*

It's that simple. You should use the lowest API level possible if you want your application to be compatible with the largest number of Android handsets. However, you must ensure that your application is tested sufficiently on any non–target platforms (any API level supported below your target SDK, as described in the next section).

### 6.4.3 Specifying the Target SDK Version :

You should always specify the `targetSdkVersion` attribute for your application. This value represents the Android SDK version your application was built for and tested against. For example, if your application was built using the APIs that are backward–compatible to Android 1.6 (API Level 4), but targeted and tested using Android 2.2 SDK (API Level 8), then you would want to specify the *targetSdkVersion* attribute as 8. Therefore, add the following to your Android manifest file within the <manifest> tag block:

*<uses–sdk android:minSdkVersion="4" android:targetSdkVersion="8" />*

Why should you specify the target SDK version you used ? Well, the Android platform has built–in functionality for backward–compatibility (to a point). Think of it like this: A specific method of a given API might have been around since API Level 1. However, the internals of that method–its behavior–might have changed slightly from SDK to SDK.

By specifying the target SDK version for your application, the Android operating system attempts to match your application with the exact version of the SDK (and the behavior as you tested it within the application), even when running a different (newer) version of the platform. This means that the application should continue to behave in "the old way" despite any new changes or "improvements" to the SDK that might cause unintended consequences in your application.

### 6.4.4 Specifying the Maximum SDK Version :

You will rarely want to specify the `maxSdkVersion` attribute for your application. This value represents the highest Android SDK version your application supports, in terms of API level. It restricts forward–compatibility of your application.

One reason you might want to set this attribute is if you want to limit who can install the application to exclude devices with the newest SDKs. For example, you might develop a free beta version of your application with plans for a paid version for the newest SDK. By setting the `maxSdkVersion` attribute of the manifest file for your free application, you disallow anyone with the newest SDK to install the free version of the application. The downside of this idea ? If your users have phones that receive over–the–air SDK updates, your application would cease to work (and appear) on phones where it had functioned perfectly, which might "upset" your users and result in bad ratings on your market of choice.

**The short answer :** Use `maxSdkVersion` only when absolutely necessary and when you understand the risks associated with its use.

## 6.5 Enforcing Application System Requirements Working with Permissions :

Android devices have different hardware and software configurations. Some devices have built–in keyboards and others rely upon the software keyboard. Similarly, certain Android devices support the latest 3–D graphics libraries and others provide little or no graphics support. The Android manifest file has several informational tags for flagging the system features and hardware configurations supported or required by an Android application.

### 6.5.1 Specifying Supported Input Methods :

The <uses–configuration> tag can be used to specify which input methods the application supports. There are different configuration attributes for five–way navigation, the hardware keyboard and keyboard types; navigation devices such as the directional pad, trackball, and wheel; and touch screen settings.

There is no "OR" support within a given attribute. If an application supports multiple input configurations, there must be multiple <uses–configuration> tags–one for each complete configuration supported.

For example, if your application requires a physical keyboard and touch screen input using a finger or a stylus, you need to define two separate <uses–configuration> tags in your manifest file, as follows:

*<uses–configuration android:reqHardKeyboard="true" android:reqTouch Screen="finger" />*

*<uses–configuration android:reqHardKeyboard="true"*

*android:reqTouchScreen="stylus" />*

For more information about the <uses–configuration> tag of the Android manifest file, see the Android SDK reference at http://developer.android.com/guide/topics/manifest/uses–configuration–element.html.

### 6.5.2 Specifying Required Device Features :

Not all Android devices support every Android feature. Put another way : There are a number of APIs (and related hardware) that Android devices may optionally include. For example, not all Android devices have multi–touch ability or a camera flash.

The <uses–feature> tag can be used to specify which Android features the application requires to run properly. These settings are for informational purposes only–the Android operating system does not enforce these settings, but publication channels such as the Android Market use this information to filter the applications available to a given user.

If your application requires multiple features, you must create a <uses–feature> tag for each. For example, an application that requires both a light and proximity sensor requires two tags :

<uses–feature android:name="android.hardware.sensor.light" />

<uses–feature android:name="android.hardware.sensor.proximity" />

One common reason to use the <*uses–feature*> tag is for specifying the OpenGL ES versions supported by your application. By default, all applications function with OpenGL ES 1.0 (which is a required feature of all Android devices). However, if your application requires features available only in later versions of OpenGL ES, such as 2.0, then you must specify this feature in the Android manifest file. This is done using the *android:glEsVersion* attribute of the <*uses–feature*> tag. Specify the lowest version of OpenGL ES that the application requires. If the application works with 1.0 and 2.0, specify the lowest version (so that the Android Market allows more users to install your application).

For more information about the <*uses–feature*> tag of the Android manifest file, see the Android SDK reference.

### 6.5.3 Specifying Supported Screen Sizes :

Android devices come in many shapes and sizes. Screen sizes and pixel densities vary widely across the range of Android devices available on the market today. The Android platform categorizes screen types in terms of sizes (small, normal, and large) and pixel density (low, medium, and high).These characteristics effectively cover the variety of screen types available within the Android platform.

An application can provide custom resources for specific screen sizes and pixel densities. The <*supportsscreen*> tag can be used to specify which Android types of screens the application supports.

For example, if the application supports QVGA screens (small) and HVGA screens (normal) regardless of pixel density, the <*supports–screen*> tag is configured as follows:

```
<supports–screens android:smallScreens="true"
android:normalScreens="true"
android:largeScreens"false"
android:anyDensity="true"/>
```

### 6.5.4 Working with External Libraries :

You can register any shared libraries your application links to within the Android manifest file. By default, every application is linked to the standard Android packages (such as android.app) and is aware of its own package. However, if your application links to additional packages, they must be registered within the <*application*> tag of the Android manifest file using the <*uses–library*> tag. For example

```
<uses–library android:name="com.sharedlibrary.sharedStuff" />
```

This feature is often used for linking to optional Google APIs. For more information about the <*uses–library*> tag of the Android manifest file, see the Android SDK reference.

## 6.6 Working with Permissions :

The Android operating system has been locked down so that applications have limited capability to adversely affect operations outside their process space. Instead, Android applications run within the bubble of their own virtual machine, with their own Linux user account (and related permissions).

### 6.6.1 Registering Permissions Your Application Requires :

Android applications have no permissions by default. Instead, any permissions for shared resources or privileged access–whether it's shared data, such as the Contacts database, or access to underlying hardware, such as the built–in camera–must be explicitly registered within the Android manifest file. These permissions are granted when the application is installed.

The following XML excerpt for the preceding Android manifest file defines a permission using the <uses–permission> tag to gain access to the built–in camera:

*<uses–permission android:name="android.permission.CAMERA" />*

A complete list of the permissions can be found in the *android. Manifest.permission* class. Your application manifest should include only the permissions required to run. The user is informed what permissions each Android application requires at install time.

### 6.6.2 Registering Permissions Your Application Grants to Other Applications :

Applications can also define their own permissions by using the *<permission>* tag. Permissions must be described and then applied to specific application components, such as Activities, using the *android:permission* attribute.

Permissions can be enforced at several points :

- When starting an Activity or Service
- When accessing data provided by a content provider
- At the function call level
- When sending or receiving broadcasts by an Intent

Permissions can have three primary protection levels: normal, dangerous, and signature. The *normal* protection level is a good default for fine–grained permission enforcement within the application. The dangerous protection level is used for higher risk Activities, which might adversely affect the device. Finally, the *signature* protection level permits any application signed with the same certificate to use that component for controlled application interoperability. You learn more about application signing in upcoming chapter.

Permissions can be broken down into categories, called permission groups, which describe or warn why specific Activities require permission. For example, permissions might be applied for Activities that expose sensitive user data such as location and personal information *(android.permission–group.LOCATION* and *android.permissiongroup.PERSONAL_INFO)*, access underlying hardware *(android.permissiongroup.HARDWARE_CONTROLS)*, or perform operations that might incur fees to the user *(android.permission–group.COST_MONEY)*. A complete list of permission groups is available within the *Manifest.permission_ group* class.

### ❑ Check Your Progress :

1. Which of the following android component displays the part of an activity on screen ?

   (A) Layout (B) Component (C) Manifest (D) XML

2. Version attributes defined within the ________ tag.

   (A) <application> (B) <manifest>

   (C) <permission> (D) <android>

3. The manifest what the application consist of glues everything together. (True/False)

   (A) True (B) False

4. To use the camera in Android Application which permission is required to set.

   (A) android.permission.CAMERA (B) android.permission.INTERNET

   (C) android.permission.CONTACT (D) android.permission.READ

5. Applications can also define their own permissions by using the ________ tag.

   (A) <android> (B) <permission> (C) <service> (D) <uses>

## 6.7 Let Us Sum Up :

In this unit we learn regarding use of Android Manifest File in Android Application, how to set the Android Manifest File in Android Application, how to implement common setting in Android Manifest File and use various permissions available in Android Application

## 6.8 Answers for Check Your Progress :

**1.** (C) **2.** (B) **3.** (A) **4.** (A) **5.** (B)

## 6.9 Glossary :

**1.** **SDK :** Software Development Kit

**2.** **AVD :** Android Virtual Device

**3.** **DDMS :** Dalvik Debug Monitor Server

**4.** **ADB :** Android Debug Bridge

## 6.10 Assignment :

1. What is AndroidManifest.xml file ? Explain it with different tags.
2. How to register the servlet in manifest file ?
3. Define AndroidManifest.xml file

## 6.11 Activities :

1. Use AndroidManifest.xml with proper tag and implementation of android program with android code.

## 6.12 Case Study :

Find the problem with AndroidManifest.xml file and try to solve it with proper justification of android code.

## 6.13 Further Reading :

Lauren Darcey and Shane Conder, "Android Wireless Application Development", Pearson Education, 2nd ed. (2011).

Unit 07

# MANAGING APPLICATION RESOURCES IN A HIERARCHY & WORKING WITH DIFFERENT TYPES OF RESOURCES

## UNIT STRUCTURE

## 7.0 Learning Objectives :

- To learn about the resources available in Android
- To manage various types of resources in Android Application
- To learn how to create and use resources in Android Application

## 7.1 Introduction :

The well–written application accesses its resources programmatically instead of hard coding them into the source code. This is done for a variety of reasons. Storing application resources in a single place is a more organized approach to development and makes the code more readable and maintainable. Externalizing resources such as strings makes it easier to localize applications for different languages and geographic regions. In this chapter, you learn how Android applications store and access important resources such as strings, graphics, and other data. You also learn how to organize Android resources within the project files for localization and different device configurations.

## 7.2 What Are Resources ?

All Android applications are composed of two things: functionality (code instructions) and data (resources). The functionality is the code that determines how your application behaves.

This includes any algorithms that make the application run. Resources include text strings, images and icons, audio files, videos, and other data used by the application.

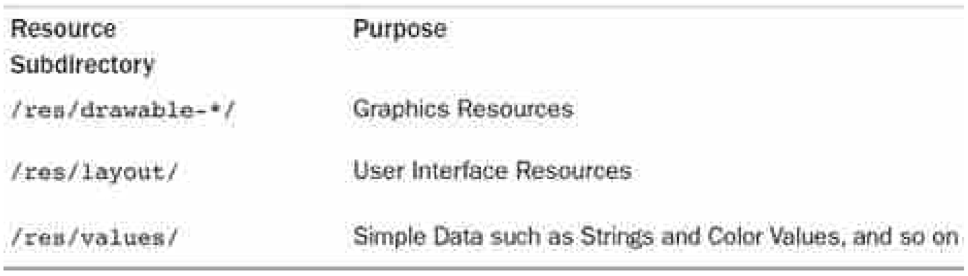
### 7.2.1 Storing Application Resources :

Android resource files are stored separately from the java class files in the Android project. Most common resource types are stored in XML. You can also store raw data files and graphics as resources.

### 7.2.2 Understanding the Resource Directory Hierarchy :

Resources are organized in a strict directory hierarchy within the Android project. All resources must be stored under the /res project directory in specially named subdirectories that must be lowercase.

Different resource types are stored in different directories. The resource sub–directories generated when you create an Android project using the Eclipse plug–in are shown in Figure 7.1.



| Resource Subdirectory | Purpose |
|---|---|
| /res/drawable-*/ | Graphics Resources |
| /res/layout/ | User Interface Resources |
| /res/values/ | Simple Data such as Strings and Color Values, and so on |

**Figure 7.1 : Default Android Resource Directories**

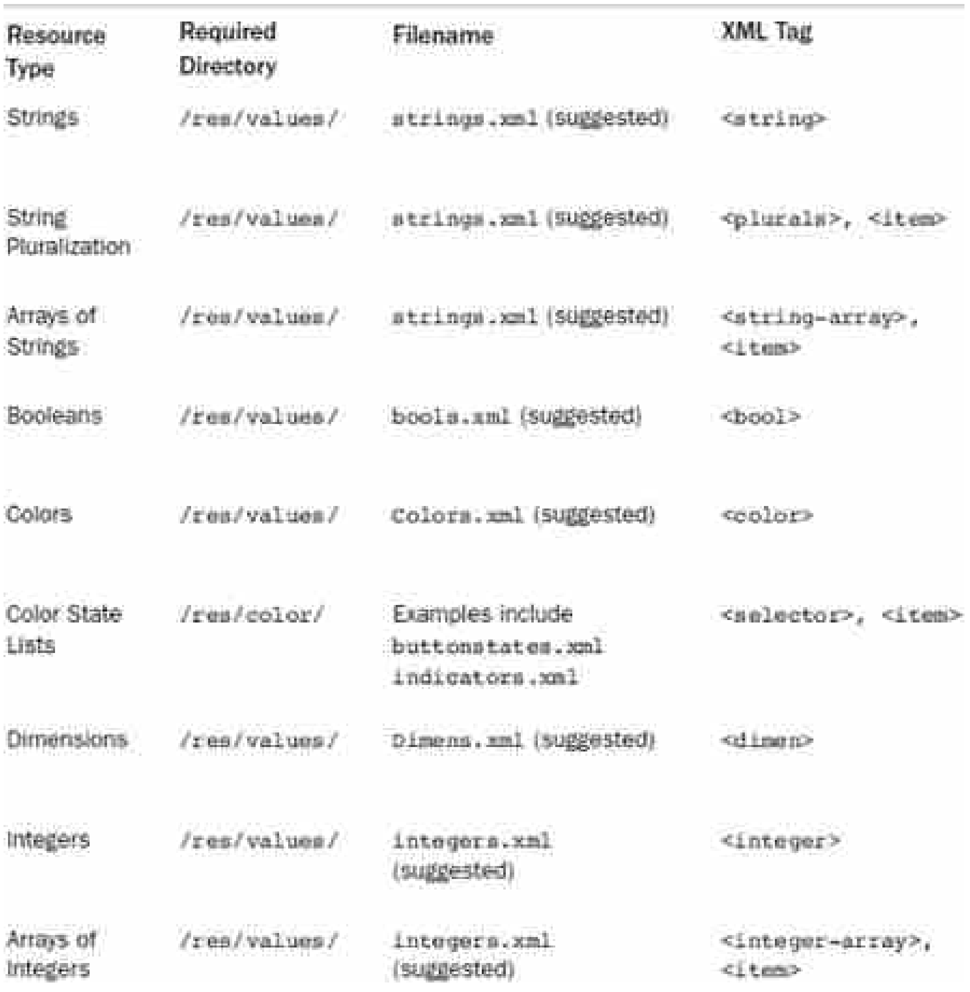
Each resource type corresponds to a specific resource subdirectory name. For example, all graphics are stored under the */res/drawable* directory structure. Resources can be further organized in a variety of ways using even more specially named directory qualifiers. For example, the */res/drawable–hdpi* directory stores graphics for high–density screens, the */res/drawable–ldpi* directory stores graphics for low–density screens, and the */res/drawable–mdpi* directory stores graphics for medium–density screens. If you had a graphic resource that was shared by all screens, you would simply store that resource in the */res/ drawable* directory. We talk more about resource directory qualifiers later in this chapter.

### 7.2.3 Using the Android Asset Packaging Tool :

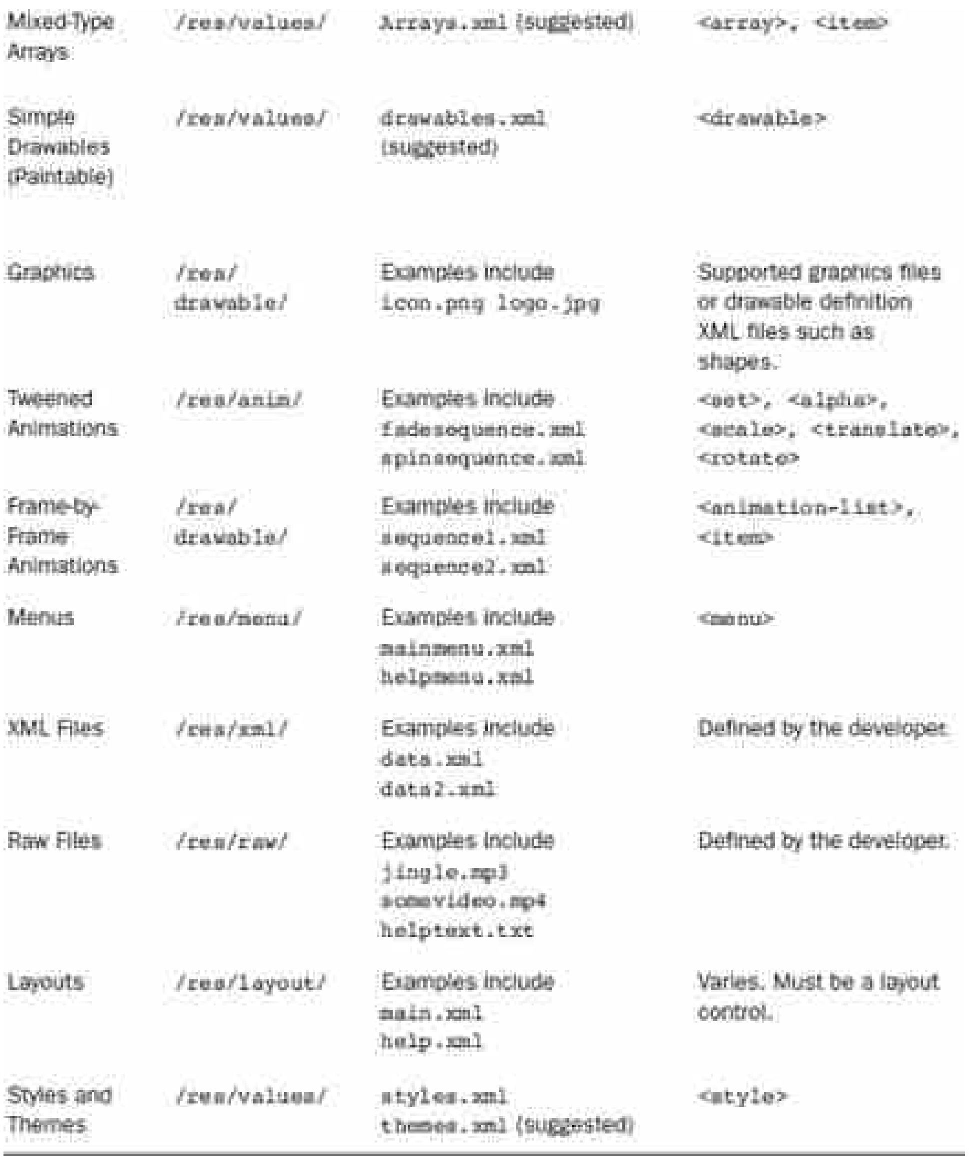
If you use the Eclipse with the Android Development Tools Plug–In, you will find that adding resources to your project is simple. The plug–in detects new resources when you add them to the appropriate project resource directory under /res automatically. These resources are compiled, resulting in the generation of the R.java file, which enables you to access your resources programmatically. If you use a different development environment, you need to use the *aapt* tool command–line interface to compile your resources and package your application binaries to deploy to the phone or emulator. You can find the *aapt* tool in the */tools* subdirectory of each specific Android SDK version.

### 7.2.4 Resource Value Types :

Android applications rely on many different types of resources–such as text strings, graphics, and color schemes–for user interface design. These resources are stored in the */res* directory of your Android project in a strict (but reasonably flexible) set of directories and files. All resources filenames must be lowercase and simple (letters, numbers, and underscores only). The resource types supported by the Android SDK and how they are stored within the project are shown in Figure 7.2.



| Resource Type | Required Directory | Filename | XML Tag |
|---|---|---|---|
| Strings | /res/values/ | strings.xml (suggested) | <string> |
| String Pluralization | /res/values/ | strings.xml (suggested) | <plurals>, <item> |
| Arrays of Strings | /res/values/ | strings.xml (suggested) | <string-array>, <item> |
| Booleans | /res/values/ | bools.xml (suggested) | <bool> |
| Colors | /res/values/ | Colors.xml (suggested) | <color> |
| Color State Lists | /res/color/ | Examples include buttonstates.xml indicators.xml | <selector>, <item> |
| Dimensions | /res/values/ | Dimens.xml (suggested) | <dimen> |
| Integers | /res/values/ | integers.xml (suggested) | <integer> |
| Arrays of Integers | /res/values/ | integers.xml (suggested) | <integer-array>, <item> |



| | | | |
|---|---|---|---|
| Mixed-Type Arrays | /res/values/ | Arrays.xml (suggested) | <array>, <item> |
| Simple Drawables (Paintable) | /res/values/ | drawables.xml (suggested) | <drawable> |
| Graphics | /res/ drawable/ | Examples include icon.png logo.jpg | Supported graphics files or drawable definition XML files such as shapes. |
| Tweened Animations | /res/anim/ | Examples include fadesequence.xml spinsequence.xml | <set>, <alpha>, <scale>, <translate>, <rotate> |
| Frame-by-Frame Animations | /res/ drawable/ | Examples include sequence1.xml sequence2.xml | <animation-list>, <item> |
| Menus | /res/menu/ | Examples include mainmenu.xml helpmenu.xml | <menu> |
| XML Files | /res/xml/ | Examples include data.xml data2.xml | Defined by the developer. |
| Raw Files | /res/raw/ | Examples include jingle.mp3 somevideo.mp4 helptext.txt | Defined by the developer. |
| Layouts | /res/layout/ | Examples include main.xml help.xml | Varies. Must be a layout control. |
| Styles and Themes | /res/values/ | styles.xml themes.xml (suggested) | <style> |

## 7.3 Storing Different Resource Value Types :

The *aapt* traverses all properly formatted files in the */res* directory hierarchy and generates the class file *R.java* in your source code directory */src* to access all variables.

Later in this chapter, we cover how to store and use each different resource type in detail, but for now, you need to understand that different types of resources are stored in different ways.

### 7.3.1 Storing Simple Resource Types Such as Strings :

Simple resource value types, such as strings, colors, dimensions, and other primitives, are stored under the ***/res/values*** project directory in XML files. Each resource file under the /res/values directory should begin with the following XML header :

*<?xml version="1.0" encoding="utf–8"?>*

Next comes the root node *<resources>* followed by the specific resource element types such as *<string>* or *<color>*. Each resource is defined using a different element name. Although the XML file names are arbitrary, the best practice is to store your resources in separate files to reflect their types, such as *strings*.xml, *colors*.xml, and so on. However, there's nothing stopping the developers from creating multiple resource files for a given type, such as two separate xml files called *bright_colors*.xml and *muted_colors*.xml, if they so choose.

**7.3.2 Storing Graphics, Animations, Menus, and Files :**

In addition to simple resource types stored in the */res/values* directory, you can also store numerous other types of resources, such as animation sequences, graphics, arbitrary XML files, and raw files. These types of resources are not stored in the */res/values* directory, but instead stored in specially named directories according to their type. For example, you can include animation sequence definitions in the */res/anim* directory. Make sure you name resource files appropriately because the resource name is derived from the filename of the specific resource. For example, a file called flag.png in the */res/drawable* directory is given the name *R.drawable.flag*.

**7.3.3 Understanding How Resources Are Resolved :**

Few applications work perfectly, no matter the environment they run in. Most require some tweaking, some special case handling. That's where alternative resources come in. You can organize Android project resources based upon more than a dozen different types of criteria, including language and region, screen characteristics, device modes (night mode, docked, and so on), input methods, and many other device differentiators. It can be useful to think of the resources stored at the top of the resource hierarchy a default resources and the specialized versions of those resources as alternative resources. Two common reasons that developers use alternative resources are for internationalization and localization purposes and to design an application that runs smoothly on different device screens and orientations.

The Android platform has a very robust mechanism for loading the appropriate resources at runtime. An example might be helpful here. Let's presume that we have a simple application with its requisite string, graphic, and layout resources. In this application, the resources are stored in the top–level resource directories (for example, /res/values/strings.xml, /res/drawable/myLogo.png, and /res/layout/main.xml). No matter what Android device (huge hi–def screen, postage–stamp–sized screen, English or Chinese language or region, portrait or landscape orientation, and so on), you run this application on, the same resource data is loaded and used. Back in our simple application example, we could create alternative string resources in Chinese simply by adding a second strings.xml file in a resource subdirectory called /res/values–zh/strings.xml (note the –zh qualifier).We could provide different logos for different screen densities by providing three versions of myLogo.png:

- /res/drawable–ldpi/myLogo.png (low–density screens)
- /res/drawable–mdpi/myLogo.png (medium–density screens)
- /res/drawable–hdpi/myLogo.png (high–density screens) Finally, let's say that the application would look much better if the layout was different in portrait versus landscape modes.We could change the layout around, moving controls around, in order to achieve a more pleasant user experience, and provide two layouts:
- /res/layout–port/main.xml (layout loaded in portrait mode)
- /res/layout–land/main.xml (layout loaded in landscape mode) With these alternative resources in place, the Android platform behaves as follows:

- If the device language setting is Chinese, the strings in /res/values–zh/ strings.xml are used. In all other cases, the strings in /res/values/strings.xml are used.
- If the device screen is a low–density screen, the graphic stored in the /res/ drawable–ldpi/myLogo.png resource directory is used. If it's a medium–density screen, the mdpi drawable is used, and so on.
- If the device is in landscape mode, the layout in the /res/layout–land/ main.xml is loaded. If it's in portrait mode, the /res/layout–port/main.xml layout is loaded.

There are four important rules to remember when creating alternative resources :

1. The Android platform always loads the most specific, most appropriate resource available. If an alternative resource does not exist, the default resource is used. Therefore, know your target devices, design for the defaults, and add alternative resources judiciously.
2. Alternative resources must always be named exactly the same as the default resources. If a string is called strHelpText in the /res/values/ strings.xml file, then it must be named the same in the /res/values–fr/ strings.xml (French) and /res/values–zh/strings.xml (Chi nese) string files.The same goes for all other types of resources, such as graphics or layout files.
3. Good application design dictates that alternative resources should always have a default counterpart so that regardless of the device, some version of the resource always loads.The only time you can get away without a default resource is when you provide every kind of alternative resource (for example, providing ldpi, mdpi, and hdpi graphics resources cover every eventuality, in theory).
4. Don't go overboard creating alternative resources, as they add to the size of your application package and can have performance implications. Instead, try to design your default resources to be flexible and scalable. For example, a good layout design can often support both landscape and portrait modes seamlessly–if you use the right controls.

### 7.3.4 Accessing Resources Programmatically :

Developers access specific application resources using the *R.java* class file and its subclasses, which are automatically generated when you add resources to your project (if you use Eclipse). You can refer to any resource identifier in your project by name. For example, the following string resource named *strHello* defined within the resource file called /res/values/strings.xml is accessed in the code as *R.string.strHello*

This variable is not the actual data associated with the string named hello. Instead, you use this resource identifier to retrieve the resource of that type (which happens to be string). For example, a simple way to retrieve the string text is to call

***String myString = getResources().getString(R.string.strHello);***

First, you retrieve the Resources instance for your application Context *(android.content.Context)*, which is, in this case, this because the Activity class extends Context. Then you use the Resources instance to get the appropriate kind of resource you want. You find that the Resources class *(android.content. res.Resources)* has helper methods for handling every kind of resource.

Before we go any further, we find it can be helpful to dig in and create some resources, so let's create a simple example. Don't worry if you don't understand every aspect of the exercise. You can find out more about each different resource type later in this chapter.

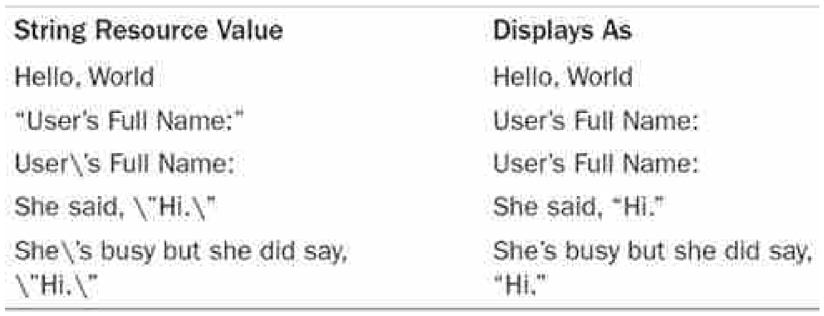
## 7.4 Working with Different Types of Resources :

In this section, we look at the specific types of resources available for Android applications, how they are defined in the project files, and how you can access this resource data programmatically.

For each type of resource type, you learn what types of values can be stored and in what format. Some resource types (such as *Strings* and *Colors*) are well supported with the Android Plug–in Resource Editor, whereas others (such as *Animation* sequences) are more easily managed by editing the XML files directly.

### 7.4.1 Working with String Resources :

String resources are among the simplest resource types available to the developer. String resources might show text labels on form views and for help text.The application name is also stored as a string resource, by default. String resources are defined in XML under the /res/values project directory and compiled into the application package at build time. All strings with apostrophes or single straight quotes need to be escaped or wrapped in double straight quotes. Some examples of well–formatted string values are shown in Figure 7.2



| String Resource Value | Displays As |
|---|---|
| Hello, World | Hello, World |
| "User's Full Name:" | User's Full Name: |
| User\'s Full Name: | User's Full Name: |
| She said, \"Hi.\" | She said, "Hi." |
| She\'s busy but she did say, \"Hi.\" | She's busy but she did say, "Hi." |

**Figure 7.2 : String Resource Formatting Examples**

You can edit the strings.xml file using the Resources tab, or you can edit the XML directly by clicking the file and choosing the *strings.xml* tab. After you save the file, the resources are automatically added to your *R.java* class file. String values are appropriately tagged with the <*string*> tag and represent a name value pair. The name attribute is how you refer to the specific string programmatically, so name these resources wisely.

Here's an example of the string resource file */res/values/strings.xml* :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">Resource Viewer</string>
<string name="test_string">Testing 1,2,3</string>
<string name="test_string2">Testing 4,5,6</string>
</resources>
```

### 7.4.2 Working with String Arrays :

You can specify lists of strings in resource files. This can be a good way to store menu options and drop–down list values. String arrays are defined in XML under the */res/values* project directory and compiled into the application package at build time.

String arrays are appropriately tagged with the <*string–array*> tag and a number of <*item*> child tags, one for each string in the array. Here's an example of a simple array resource file */res/values/arrays.xml* :

```
    <?xml version="1.0" encoding="utf-8"?>
    <resources>
<string-array name="flavors">
<item>Vanilla Bean</item>
<item>Chocolate Fudge Brownie</item>
<item>Strawberry Cheesecake</item>
<item>Coffee, Coffee, Buzz Buzz Buzz</item>
<item>Americone Dream</item>
</string-array>
<string-array name="soups">
<item>Vegetable minestrone</item>
<item>New England clam chowder</item>
<item>Organic chicken noodle</item>
</string-array>
</resources>
```

As shown earlier in this chapter, accessing string arrays resources is easy. The following code retrieves a string array named flavors :

```
String[] aFlavors =
getResources().getStringArray(R.array.flavors);
```

### 7.4.3 Working with *Boolean* Resources :

Other primitive types are supported by the Android resource hierarchy as well. Boolean resources can be used to store information about application game preferences and default values. Boolean resources are defined in XML under the */res/values* project directory and compiled into the application package at build time.

**Defining Boolean Resources in XML**

Boolean values are appropriately tagged with the <bool> tag and represent a name–value pair. The name attribute is how you refer to the specific Boolean value programmatically, so name these resources wisely.

Here's an example of the Boolean resource file /res/values/bools.xml :

```
<?xml version="1.0" encoding="utf–8"?>
<resources>
<bool name="bOnePlusOneEqualsTwo">true</bool>
<bool name="bAdvancedFeaturesEnabled">false</bool>
</resources>
```

**Using Boolean Resources Programmatically**

To use a Boolean resource, you must load it using the Resource class. The following code accesses your application's Boolean resource named bAdvancedFeaturesEnabled.

```
boolean bAdvancedMode =
getResources().getBoolean(R.bool.bAdvancedFeaturesEnabled);
```

### 7.4.4 Working with *Integer* Resources :

In addition to strings and Boolean values, you can also store integers as resources. Integer resources are defined in XML under the /res/values project directory and compiled into the application package at build time.

**Defining Integer Resources in XML**

Integer values are appropriately tagged with the <integer> tag and represent a name value pair. The name attribute is how you refer to the specific integer programmatically, so name these resources wisely.

Here's an example of the integer resource file */res/values/nums.xml* :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<integer name="numTimesToRepeat">25</integer>
<integer name="startingAgeOfCharacter">3</integer>
</resources>
```

**Using Integer Resources Programmatically**

To use the integer resource, you must load it using the Resource class. The following code accesses your application's integer resource named numTimesToRepeat :

```
int repTimes =
getResources().getInteger(R.integer.numTimesToRepeat);
```

### 7.4.5 Working with *Colors* :

Android applications can store RGB color values, which can then be applied to other screen elements. You can use these values to set the color of text or other elements, such as the screen background. Color resources are defined in XML under the /res/values project directory and compiled into the application package at build time.

RGB color values always start with the hash symbol (#).The alpha value can be given for transparency control.The following color formats are supported:

- #RGB (example, #F00 is 12–bit color, red)
- #ARGB (example, #8F00 is 12–bit color, red with alpha 50%)
- #RRGGBB (example, #FF00FF is 24–bit color, magenta)
- #AARRGGBB (example, #80FF00FF is 24–bit color, magenta with alpha 50%)

Color values are appropriately tagged with the <color> tag and represent a name–value pair.

Here's an example of a simple color resource file /res/values/colors.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<color name="background_color">#006400</color>
<color name="text_color">#FFE4C4</color>
</resources>
```

The example at the beginning of the chapter accessed a color resource. Color resources are simply integers. The following code retrieves a color resource called `prettyTextColor` :

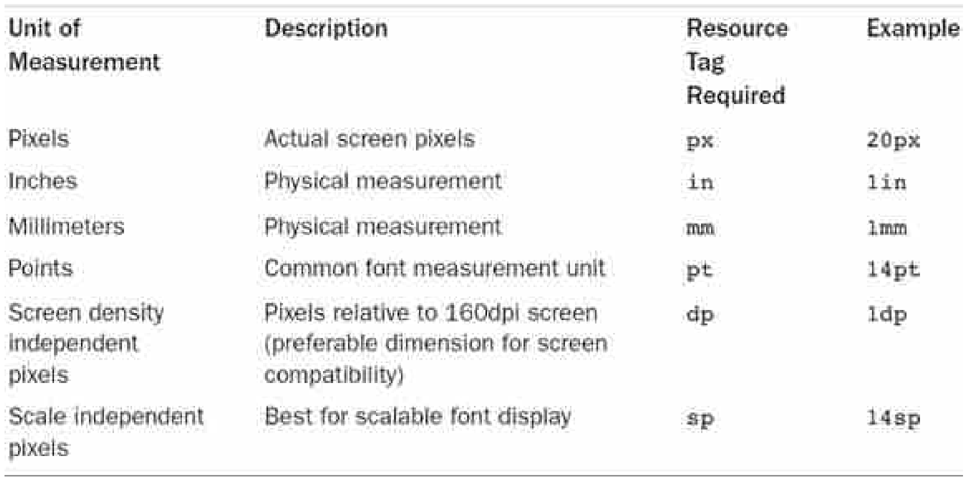
```
int myResourceColor =
getResources().getColor(R.color.prettyTextColor);
```

### 7.4.6 Working with *Dimensions* :

Many user interface layout controls such as text controls and buttons are drawn to specific dimensions. These dimensions can be stored as resources. Dimension values always end with a unit of measurement tag.

Dimension values are appropriately tagged with the <*dimen*> tag and represent a name value pair. Dimension resources are defined in XML under the /res/values project directory and compiled into the application package at build time.

The dimension units supported are shown in Figure 7.3



| Unit of Measurement | Description | Resource Tag Required | Example |
|---|---|---|---|
| Pixels | Actual screen pixels | `px` | `20px` |
| Inches | Physical measurement | `in` | `1in` |
| Millimeters | Physical measurement | `mm` | `1mm` |
| Points | Common font measurement unit | `pt` | `14pt` |
| Screen density independent pixels | Pixels relative to 160dpi screen (preferable dimension for screen compatibility) | `dp` | `1dp` |
| Scale independent pixels | Best for scalable font display | `sp` | `14sp` |

**Figure 7.3 : Dimension Unit Measurements Supported**

Here's an example of a simple dimension resource file /res/values/dimens.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<dimen name="FourteenPt">14pt</dimen>
<dimen name="OneInch">1in</dimen>
<dimen name="TenMillimeters">10mm</dimen>
<dimen name="TenPixels">10px</dimen>
</resources>
```

Dimension resources are simply floating point values. The following code retrieves a dimension resource called textPointSize :

```
float myDimension =
getResources().getDimension(R.dimen.textPointSize);
```

### 7.4.7 Working with Simple Drawables :

You can specify simple colored rectangles by using the drawable resource type, which can then be applied to other screen elements. These drawable types are defined in specific paint colors, much like the Color resources are defined.

Simple paintable drawable resources are defined in XML under the */res/values* project directory and compiled into the application package at build time. Paintable drawable resources use the <drawable> tag and represent a name–value pair.

Here's an example of a simple drawable resource file */res/values/drawables.xml* :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<drawable name="red_rect">#F00</drawable>
</resources>
```

Although it might seem a tad confusing, you can also create XML files that describe other Drawable subclasses, such as ShapeDrawable. Drawable XML definition files are stored in the /res/drawable directory within your project along with image files. This is not the same as storing <drawable> resources, which are paintable drawables. PaintableDrawable resources are stored in the /res/values directory, as explained in the previous section.

Here's a simple ShapeDrawable described in the file res/drawable/red_oval.xml :

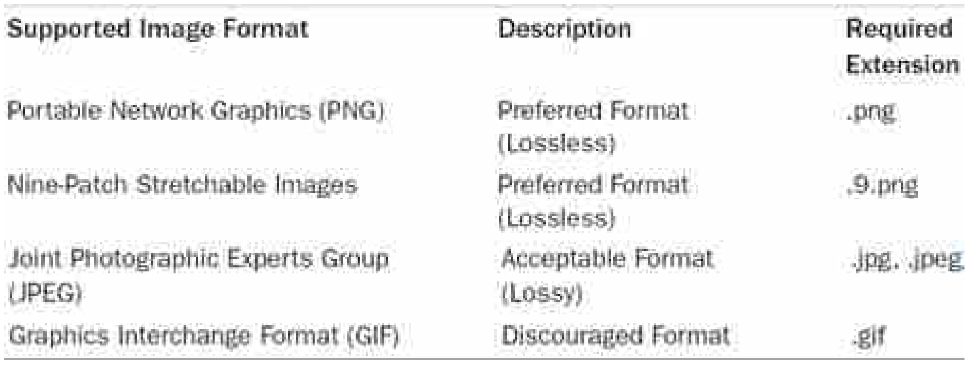
```
<?xml version="1.0" encoding="utf-8"?>
<shape
xmlns:android=
"http://schemas.android.com/apk/res/android"
android:shape="oval">
<solid android:color="#f00"/>
</shape>
```

Drawable resources defined with <drawable> are simply rectangles of a given color, which is represented by the Drawable subclass ColorDrawable.The following code retrieves a *ColorDrawable* resource called *redDrawable* :

```
import android.graphics.drawable.ColorDrawable;
...
ColorDrawable myDraw =
(ColorDrawable)getResources().
getDrawable(R.drawable.redDrawable);
```

### 7.4.8 Working with *Images* :

Applications often include visual elements such as icons and graphics. Android supports several image formats that can be directly included as resources for your application. These image formats are shown in Figure 7.4.



| Supported Image Format | Description | Required Extension |
|---|---|---|
| Portable Network Graphics (PNG) | Preferred Format (Lossless) | .png |
| Nine-Patch Stretchable Images | Preferred Format (Lossless) | .9.png |
| Joint Photographic Experts Group (JPEG) | Acceptable Format (Lossy) | .jpg, .jpeg |
| Graphics Interchange Format (GIF) | Discouraged Format | .gif |

**Figure 7.4 : Image Formats Supported in Android**

**7.4.9 Working with Animation :**

Android supports frame–by–frame animation and tweening. Frame–by–frame animation involves the display of a sequence of images in rapid succession.Tweened animation involves applying standard graphical transformations such as rotations and fades upon a single image.

The Android SDK provides some helper utilities for loading and using animation resources. These utilities are found in the *android.view.animation. AnimationUtils* class.

**Defining and Using Frame–by–Frame Animation Resources**

Frame–by–frame animation is often used when the content changes from frame to frame.

This type of animation can be used for complex frame transitions–much like a kid's flip–book.

To define frame–by–frame resources, take the following steps :

1. Save each frame graphic as an individual drawable resource. It may help to name your graphics sequentially, in the order in which they are displayed–for example, frame1.png, frame2.png, and so on.
2. Define the animation set resource in an XML file within /res/drawable/ resource directory.
3. Load, start, and stop the animation programmatically

Here's an example of a simple frame–by–frame animation resource file *\/res/drawable/juggle.xml* that defines a simple three–frame animation that takes 1.5 seconds :

```
<?xml version="1.0" encoding="utf-8" ?>
<animation-list
xmlns:android=
"http://schemas.android.com/apk/res/android"
android:oneshot="false">
<item
android:drawable="@drawable/splash1"
android:duration="50" />
<item
android:drawable="@drawable/splash2"
```

```
android:duration="50" />
<item
android:drawable="@drawable/splash3"
android:duration="50" />
</animation-list>
```

Frame–by–frame animation set resources defined with <animation–list> are represented by the Drawable subclass AnimationDrawable. The following code retrieves an Animation–Drawable resource called juggle:

```
import android.graphics.drawable.AnimationDrawable;
...
AnimationDrawable jugglerAnimation =
(AnimationDrawable)getResources().
getDrawable(R.drawable.juggle);
```

After you have a valid AnimationDrawable, you can assign it to a View on the screen and use the Animation methods to start and stop animation.

**Defining and Using Tweened Animation Resources**

Tweened animation features include scaling, fading, rotation, and translation. These actions can be applied simultaneously or sequentially and might use different interpolators.

Frame–by–frame animation is often used when the content changes from frame to frame.

This type of animation can be used for complex frame transitions–much like a kid's flip–book.

To define frame–by–frame resources, take the following steps :

1. Save each frame graphic as an individual drawable resource. It may help to name your graphics sequentially, in the order in which they are displayed–for example, frame1.png, frame2.png, and so on.
2. Define the animation set resource in an XML file within /res/drawable/ resource directory.
3. Load, start, and stop the animation programmatically.

Here's an example of a simple frame–by–frame animation resource file /res/drawable/juggle.xml that defines a simple three–frame animation that takes 1.5 seconds :

```
<?xml version="1.0" encoding="utf-8" ?>
<animation-list
xmlns:android=
"http://schemas.android.com/apk/res/android"
android:oneshot="false">
<item
android:drawable="@drawable/splash1"
android:duration="50" />
<item
```

```
android:drawable="@drawable/splash2"
android:duration="50" />
<item
android:drawable="@drawable/splash3"
android:duration="50" />
</animation-list>
```

Frame–by–frame animation set resources defined with <animation–list> are represented by the Drawable subclass AnimationDrawable.The following code retrieves an Animation–Drawable resource called juggle :

```
import android.graphics.drawable.AnimationDrawable;
...
AnimationDrawable jugglerAnimation =
(AnimationDrawable)getResources().
getDrawable(R.drawable.juggle);
```

After you have a valid AnimationDrawable, you can assign it to a View on the screen and use the Animation methods to start and stop animation.

**Defining and Using Tweened Animation Resources**

Tweened animation features include scaling, fading, rotation, and translation. These actions can be applied simultaneously or sequentially and might use different interpolators.

Tweened animation sequences are not tied to a specific graphic file, so you can write one sequence and then use it for a variety of different graphics. For example, you can make moon, star, and diamond graphics all pulse using a single scaling sequence, or you can make them spin using a rotate sequence.

Graphic animation sequences can be stored as specially formatted XML files in the /res/anim directory and are compiled into the application binary at build time.

Here's an example of a simple animation resource file /res/anim/spin.xml that defines a simple rotate operation–rotating the target graphic counterclockwise four times in place, taking 10 seconds to complete :

```
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android
="http://schemas.android.com/apk/res/android"
android:shareInterpolator="false">
<set>
<rotate
android:fromDegrees="0"
android:toDegrees="-1440"
android:pivotX="50%"
android:pivotY="50%"
android:duration="10000" />
</set>
</set>
```

If we go back to the example of a BitmapDrawable earlier, we can now add some animation simply by adding the following code to load the animation resource file spin.xml and set the animation in motion:

```
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.ImageView;
...
ImageView flagImageView =
(ImageView)findViewById(R.id.ImageView01);
flagImageView.setImageResource(R.drawable.flag);
...
Animation an =
AnimationUtils.loadAnimation(this, R.anim.spin);
flagImageView.startAnimation(an);
```

Now you have your graphic spinning. Notice that we loaded the animation using the base class object Animation. You can also extract specific animation types using the subclasses that match: RotateAnimation, ScaleAnimation, TranslateAnimation, and AlphaAnimation.

There are a number of different interpolators you can use with your tweened animation sequences.

### 7.4.10 Working with *Menus* :

Tweened animation sequences are not tied to a specific graphic file, so you can write one sequence and then use it for a variety of different graphics. For example, you can make moon, star, and diamond graphics all pulse using a single scaling sequence, or you can make them spin using a rotate sequence.

Graphic animation sequences can be stored as specially formatted XML files in the */res/anim* directory and are compiled into the application binary at build time.

Here's an example of a simple animation resource file */res/anim/spin.xml* that defines a simple rotate operation–rotating the target graphic counterclockwise four times in place, taking 10 seconds to complete :

```
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android
="http://schemas.android.com/apk/res/android"
android:shareInterpolator="false">
<set>
<rotate
android:fromDegrees="0"
android:toDegrees="-1440"
android:pivotX="50%"
android:pivotY="50%"
android:duration="10000" />
```

```
</set>
</set>
```

If we go back to the example of a BitmapDrawable earlier,we can now add some animation simply by adding the following code to load the animation resource file spin.xml and set the animation in motion:

```
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.ImageView;
...
ImageView flagImageView =
(ImageView)findViewById(R.id.ImageView01);
flagImageView.setImageResource(R.drawable.flag);
...
Animation an =
AnimationUtils.loadAnimation(this, R.anim.spin);
flagImageView.startAnimation(an);
```

Now you have your graphic spinning. Notice that we loaded the animation using the base class object Animation.You can also extract specific animation types using the subclasses that match : RotateAnimation, ScaleAnimation, TranslateAnimation, and AlphaAnimation.

There are a number of different interpolators you can use with your tweened animation sequences.

To access the preceding menu resource called /res/menu/speed.xml, simply override the method onCreateOptionsMenu() in your application :

```
public boolean onCreateOptionsMenu(Menu menu) {
getMenuInflater().inflate(R.menu.speed, menu);
return true;
}
```

### 7.4.11 Working with *XML* Files :

You can include arbitrary XML resource files to your project.You should store these XML files in the */res/xml* directory, and they are compiled into the application package at build time.

The Android SDK has a variety of packages and classes available for XML manipulation.

For now, we create an XML resource file and access it through code.

First, put a simple XML file in /res/xml directory. In this case, the file my_pets.xml with the following contents can be created :

```
<?xml version="1.0" encoding="utf-8"?>
<pets>
<pet name="Bit" type="Bunny" />
<pet name="Nibble" type="Bunny" />
<pet name="Stack" type="Bunny" />
```

```
<pet name="Queue" type="Bunny" />
<pet name="Heap" type="Bunny" />
<pet name="Null" type="Bunny" />
<pet name="Nigiri" type="Fish" />
<pet name="Sashimi II" type="Fish" />
<pet name="Kiwi" type="Lovebird" />
</pets>
```

Now you can access this XML file as a resource programmatically in the following manner :

```
XmlResourceParser myPets =
    getResources().getXml(R.xml.my_pets);
```

Finally, to prove this is XML, here's one way you might churn through the XML and extract the information :

```
import org.xmlpull.v1.XmlPullParserException;
import android.content.res.XmlResourceParser;
...
int eventType = -1;
while (eventType != XmlResourceParser.END_DOCUMENT) {
if(eventType == XmlResourceParser.START_DOCUMENT) {
Log.d(DEBUG_TAG, "Document Start");
} else if(eventType == XmlResourceParser.START_TAG) {
String strName = myPets.getName();
if(strName.equals("pet")) {
Log.d(DEBUG_TAG, "Found a PET");
Log.d(DEBUG_TAG,
"Name: "+myPets.
getAttributeValue(null, "name"));
Log.d(DEBUG_TAG,
"Species: "+myPets.
getAttributeValue(null, "type"));
}
}
eventType = myPets.next();
}
    Log.d(DEBUG_TAG, "Document End");
```

### 7.4.12 Working with *Raw* Files :

Your application can also include raw files as part of its resources. For example, your application might use raw files such as audio files, video files, and other file formats not supported by the Android Resource packaging tool aapt.

All raw resource files are included in the */res/raw* directory and are added to your package without further processing.

The resource filename must be unique to the directory and should be descriptive because the filename (without the extension) becomes the name by which the resource is accessed.

You can access raw file resources and any resource from the */res/ drawable* directory (bitmap graphics files, anything not using the <resource> XML definition method).

Here's one way to open a file called the_help.txt :

```
import java.io.InputStream;
...
InputStream iFile =
getResources().openRawResource(R.raw.the_help);
```

**7.4.13 Working with *Layouts* :**

Much as web designers use HTML, user interface designers can use XML to define Android application screen elements and layout.A layout XML resource is where many different resources come together to form the definition of an Android application screen.

Layout resource files are included in the /res/layout/ directory and are compiled into the application package at build time. Layout files might include many user interface controls and define the layout for an entire screen or describe custom controls used in other layouts.

Here's a simple example of a layout file (/res/layout/main.xml) that sets the screen's background color and displays some text in the middle of the screen (see Figure 7.3).

The main.xml layout file that displays this screen references a number of other resources, including colors, strings, and dimension values, all of which were defined in the strings.xml, colors.xml, and dimens.xml resource files.The color resource for the screen background color and resources for a TextView control's color, string, and text size 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="@color/background_color">
<TextView
android:id="@+id/TextView01"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:text="@string/test_string"
android:textColor="@color/text_color"
```

```
    android:gravity="center"
    android:textSize="@dimen/text_size"></TextView>
        </LinearLayout>
```

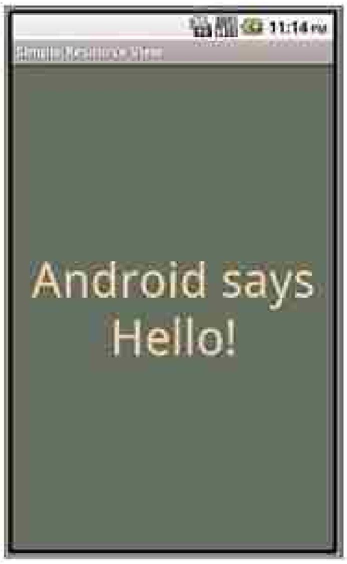


**Figure 7.5 : How the main.xml layout file displays in the emulator**

The preceding layout describes all the visual elements on a screen. In this example, a LinearLayout control is used as a container for other user interface controls–here, a single TextView that displays a line of text.

### 7.4.14 Working with *Styles* :

Android user interface designers can group layout element attributes together in styles. Layout controls are all derived from the View base class, which has many useful attributes. Individual controls, such as *Checkbox*, *Button*, and *TextView*, have specialized attributes associated with their behavior.

Styles are tagged with the <*style*> tag and should be stored in the / res/values/ directory.

Style resources are defined in XML and compiled into the application binary at build time.

Here's an example of a simple style resource file /res/values/styles.xml containing two styles : one for mandatory form fields, and one for optional form fields on TextView and EditText objects :

```
    <?xml version="1.0" encoding="utf-8"?>
    <resources>
    <style name="mandatory_text_field_style">
    <item name="android:textColor">#000000</item>
    <item name="android:textSize">14pt</item>
    <item name="android:textStyle">bold</item>
    </style>
    <style name="optional_text_field_style">
    <item name="android:textColor">#0F0F0F</item>
    <item name="android:textSize">12pt</item>
    <item name="android:textStyle">italic</item>
```

```
</style>
</resources>
```

Many useful style attributes are colors and dimensions. It would be more appropriate to use references to resources. Here's the styles.xml file again; this time, the color and text size fields are available in the other resource files colors.xml and dimens.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<style name="mandatory_text_field_style">
<item name="android:textColor"
>@color/mand_text_color</item>
<item name="android:textSize"
>@dimen/important_text</item>
<item name="android:textStyle">bold</item>
</style>
<style name="optional_text_field_style">
<item name="android:textColor"
>@color/opt_text_color</item>
<item name="android:textSize"
>@dimen/unimportant_text</item>
<item name="android:textStyle">italic</item>
</style>
</resources>
```

Now, if you can create a new layout with a couple of TextView and EditText text controls, you can set each control's style attribute by referencing it as such :

style="@style/name_of_style"

Here we have a form layout called /res/layout/form.xml that does that:

```
<?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="@color/background_color">
<TextView
android:id="@+id/TextView01"
style="@style/mandatory_text_field_style"
android:layout_height="wrap_content"
android:text="@string/mand_label"
```

```
android:layout_width="wrap_content" />
<EditText
android:id="@+id/EditText01"
style="@style/mandatory_text_field_style"
android:layout_height="wrap_content"
android:text="@string/mand_default"
android:layout_width="fill_parent"
android:singleLine="true" />
<TextView
android:id="@+id/TextView02"
style="@style/optional_text_field_style"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/opt_label" />
<EditText
android:id="@+id/EditText02"
style="@style/optional_text_field_style"
android:layout_height="wrap_content"
android:text="@string/opt_default"
android:singleLine="true"
android:layout_width="fill_parent" />
<TextView
android:id="@+id/TextView03"
style="@style/optional_text_field_style"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/opt_label" />
<EditText android:id="@+id/EditText03"
style="@style/optional_text_field_style"
android:layout_height="wrap_content"
android:text="@string/opt_default"
android:singleLine="true"
android:layout_width="fill_parent" />
</LinearLayout>
```

The resulting layout has three fields, each made up of one TextView for the label and one EditText where the user can input text. The mandatory style is applied to the mandatory label and text entry. The other two fields use the optional style. The resulting layout would look something like Figure 7.6.

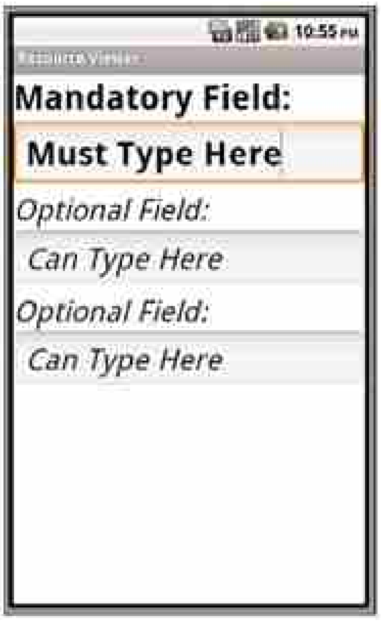


**Figure 7.6 : A layout using two styles, one for mandatory fields and another for optional fields.**

❑ **Check Your Progress :**

1. AAPT stands for ________

   (A) Android Asset Packaging Tool

   (B) Asset Android Packaging Tool

   (C) Application Asset Packaging Tool

   (D) Android Application Packaging Tool

2. What is the nine–patch images tool in android ?

   (A) Image

   (B) It is used to change the bitmap images into nine sections

   (C) Tools

   (D) None of the Above

3. Which media format is not supported by Android ?

   (A) jpeg (B) bmp (C) AVI (D) jpg

4. In which directory XML Layout files are stored ?

   (A) asset (B) src (C) drawing (D) res/layout

5. Android user interface designers can group layout element attributes together in ________.

   (A) Styles (B) Android (C) Manifest (D) XML

## 7.5 Let Us Sum Up :

In this unit we learn regarding various android resources available, use of various resources and application of various resources.

## 7.6 Answers for Check Your Progress :

**1.** (A) **2.** (B) **3.** (C) **4.** (D) **5.** (A)

## 7.7 Glossary :

1. **SDK :** Software Development Kit
2. **AVD :** Android Virtual Device
3. **DDMS :** Dalvik Debug Monitor Server

4 ADB: Android Debug Bridge

## 7.8 Assignment :

1. What are the Resources ? Explain Working with different resources with required directory filename and xml tag
2. What is Animation ? Explain Frame–by–Frame and Tweened Animation in details
3. Explain the following color formats. #RGB, #ARGB, #RRGGBB & #AARRGGBB

## 7.9 Activities :

1. Understand the use of various resources available in the android with proper application identification and to implement the same with use of the resources.

## 7.10 Case Study :

Identify the c real–life application and implement the resources in it with the proper justification and use.

## 7.11 Further Reading :

Lauren Darcey and Shane Conder, "Android Wireless Application Development", Pearson Education, 2nd ed. (2011).

## BLOCK SUMMARY :

In this block we have discussed regarding to understand the anatomy of Android application, to understand the Android Terminologies, to understand the Application Context, Activities, Services & Intents, to understand how to receive and broadcast the intents, to use Android Manifest file and understand its common settings, to use and set the various android permissions, and to understand how to Manage Application resources in a hierarchy & Working with different types of resources.

**BLOCK ASSIGNMENT :**

1. What is the role of ADT in android platform ?
2. What is android emulato r?
3. Explain the building blocks of Android Application.
4. Discuss the life cycle of Android Activity in details
5. What is Activity ?
6. Define: Context
7. How to create Intent in Android ?
8. Define: Service
9. What is Android ? Explain various components of Android in details
10. Define Activity ? Explain life cycle methods of android activity.
11. How to create intent in android? Explain with suitable example.
12. What are the Resources ? Explain Working with different resources with required directory filename and xml tag
13. What is Animation ? Explain Frame–by–Frame and Tweened Animation in details
14. Explain the following color formats. #RGB, #ARGB, #RRGGBB & #AARRGGBB

❖ **Short Questions :**

1. What is android emulator ?
2. What is Activity ?
3. Define: Context
4. How to create Intent in Android ?
5. Define: Service
6. What is the role of ADT in android platform ?

❖ **Long Questions :**

1. What is Android ? Explain various components of Android in details
2. Define Activity ? Explain life cycle methods of android activity.
3. How to create intent in android ? Explain with suitable example.
4. What are the Resources ? Explain Working with different resources with required directory filename and xml tag
5. What is Animation ? Explain Frame–by–Frame and Tweened Animation in details
6. Explain the following color formats. #RGB, #ARGB, #RRGGBB & #AARRGGBB

❖ **Enrolment No. :** ☐

1. How many hours did you need for studying the units ?

| Unit No. | 4 | 5 | 6 | 7 |
|---|---|---|---|---|
| No. of Hrs. | | | | |

2. Please give your reactions to the following items based on your reading of the block :

| Items | Excellent | Very Good | Good | Poor | Give specific example if any |
|---|---|---|---|---|---|
| Presentation Quality | ☐ | ☐ | ☐ | ☐ | ____ |
| Language and Style | ☐ | ☐ | ☐ | ☐ | ____ |
| Illustration used (Diagram, tables etc) | ☐ | ☐ | ☐ | ☐ | ____ |
| Conceptual Clarity | ☐ | ☐ | ☐ | ☐ | ____ |
| Check your progress Quest | ☐ | ☐ | ☐ | ☐ | ____ |
| Feed back to CYP Question | ☐ | ☐ | ☐ | ☐ | ____ |

3. Any other Comments

..........................................................................................................

..........................................................................................................

..........................................................................................................

..........................................................................................................

..........................................................................................................

..........................................................................................................

..........................................................................................................

..........................................................................................................

Dr. Babasaheb Ambedkar Open University Ahmedabad

BCAR-503

# ***MOBILE APPLICATION DEVELOPMENT (USING ANDROID)***

**BLOCK 3 : ANDROID USER INTERFACE DESIGN ESSENTIALS**

# *ANDROID USER INTERFACE DESIGN ESSENTIALS*

## Block Introduction :

Most Android applications inevitably need some form of user interface. In this chapter, we discuss the user interface elements available within the Android Software Development Kit (SDK). Some of these elements display information to the user, whereas others gather information from the user. You learn how to use a variety of different components and controls to build a screen and how your application can listen for various actions performed by the user. Finally, you learn how to style controls and apply themes to entire screens.

## Block Objectives :

- To understand the user interface screen elements
- To design the user interface and layouts
- To understand working with Drawing and Amination

## Block Structure :

Unit 08

# USER INTERFACE SCREEN ELEMENTS

## UNIT STRUCTURE

## 8.0 Learning Objectives :

- To learn how to use interface & Android control
- To learn how to create screen elements with control
- To understand the views and layouts

## 8.1 Introduction :

Before we go any further, we need to define a few terms. This gives you a better understanding of certain capabilities provided by the Android SDK before they are fully introduced. First, let's talk about the View and what it is to the Android SDK.

The Android SDK has a Java packaged named android.view. This package contains a number of interfaces and classes related to drawing on the screen. However, when we refer to the View object, we actually refer to only one of the classes within this package: the android.view.View class. The View class is the basic user interface building block within Android. It represents a rectangular portion of the screen. The View class serves as the base class for nearly all the user interface controls and layouts within the Android SDK

## 8.2 Introducing the Android Control :

The Android SDK contains a Java package named android.widget. When we refer to controls, we are typically referring to a class within this package. The Android SDK includes classes to draw most common objects, including ImageView, FrameLayout, EditText, and Button classes. As mentioned previously, all controls are typically derived from the View class. This chapter is primarily about controls that display and collect data from the user. We cover many of these basic controls in detail.

## 8.3 Introducing the Android Layout :

One special type of control found within the android.widget package is called a layout. A layout control is still a View object, but it doesn't actually draw anything specific on the screen. Instead, it is a parent container for organizing other controls (children). Layout controls determine how and where on the screen child controls are drawn. Each type of layout control draws its children using particular rules. For instance, the LinearLayout control draws its child controls in a single horizontal row or a single vertical column. Similarly, a TableLayout control displays each child control in tabular format (in cells within specific rows and columns).

These special View controls, which are derived from the android.view. ViewGroup class, are useful only after you understand the various display controls these containers can hold. By necessity, we use some of the layout View objects within this chapter to illustrate how to use the controls previously mentioned. However,we don't go into the details of the various layout types available as part of the Android SDK until the next chapter.

## 8.4 Displaying Text to Users with TextView :

One of the most basic user interface elements, or controls, in the Android SDK is the TextView control. You use it, quite simply, to draw text on the screen.You primarily use it to display fixed text strings or labels.

Frequently, the TextView control is a child control within other screen elements and controls. As with most of the user interface elements, it is derived from View and is within the android.widget package. Because it is a View, all the standard attributes such as width, height, padding, and visibility can be applied to the object. However, as a text–displaying control, you can apply many other TextView–specific attributes to control behavior and how the text is viewed in a variety of situations.

First, though, let's see how to put some quick text up on the screen. <TextView> is the XML layout file tag used to display text on the screen.You can set the android:text property of the TextView to be either a raw text string in the layout file or a reference to a string resource.

Here are examples of both methods you can use to set the android:text attribute of a TextView.The first method sets the text attribute to a raw string; the second method uses a string resource called sample_text, which must be defined in the strings.xml resource file.

```
<TextView
android:id="@+id/TextView01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Some sample text here" />
<TextView
android:id="@+id/TextView02"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/sample_text" />
```

To display this TextView on the screen, all your Activity needs to do is call the setContentView() method with the layout resource identifier in which you defined the preceding XML shown.You can change the text displayed programmatically by calling the setText() method on the TextView object. Retrieving the text is done with the getText() method.

Now let's talk about some of the more common attributes of TextView objects.

### 8.4.1 Configuring Layout and Sizing :

The TextView control has some special attributes that dictate how the text is drawn and flows.You can, for instance, set the TextView to be a single line high and a fixed width. If, however, you put a long string of text that can't fit, the text truncates abruptly. Luckily, there are some attributes that can handle this problem. The width of a TextView can be controlled in terms of the ems measurement rather than in ixels.An em is a term used in typography that is defined in terms of the point size of a particular font. (For example, the measure of an em in a 12–point font is 12 points.)

This measurement provides better control over how much text is viewed, regardless of the font size. Through the ems attribute, you can set the width of the TextView.Additionally, you can use the maxEms and minEms attributes to set the maximum width and minimum width, respectively, of the TextView in terms of ems.

The height of a TextView can be set in terms of lines of text rather than pixels.Again, this is useful for controlling how much text can be viewed regardless of the font size. The lines attribute sets the number of lines that the TextView can display.You can also use maxLines and minLines to control the maximum height and minimum height, respectively, that the Textview displays.

Here is an example that combines these two types of sizing attributes.This TextView is two lines of text high and 12 ems of text wide.The layout width and height are specified to the size of the TextView and are required attributes in the XML schema :

```
<TextView
android:id="@+id/TextView04"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:lines="2"
android:ems="12"
android:text="@string/autolink_test" />
```

Instead of having the text only truncate at the end, as happens in the preceding example, we can enable the ellipsize attribute to replace the last couple characters with an ellipsis (...) so the user knows that not all text is displayed.

### 8.4.2 Creating Contextual Links in Text :

If your text contains references to email addresses,web pages, phone numbers, or even street addresses, you might want to consider using the attribute autoLink (see Figure 8.1).

The autoLink attribute has four values that you can use in combination with each other.

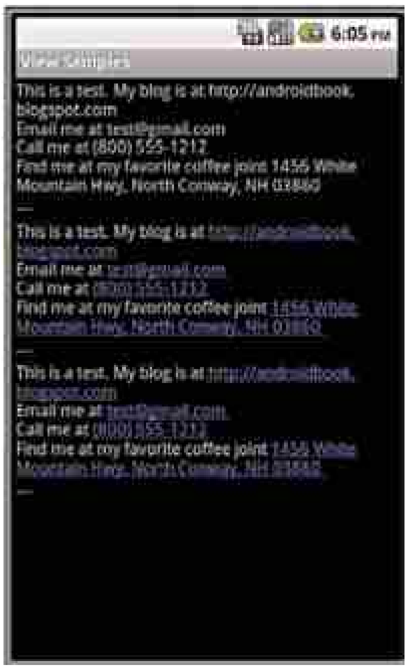
When enabled, these autoLink attribute values create standard web–style links to the application that can act on that data type. For instance, setting the attribute to web automatically finds and links any URLs to web pages.

Your text can contain the following values for the autoLink attribute :

- **none :** Disables all linking.
- **web :** Enables linking of URLs to web pages.
- **email :** Enables linking of email addresses to the mail client with the recipient filled.

- **phone :** Enables linking of phone numbers to the dialer application with the phone number filled out, ready to be dialed.
- **map :** Enables linking of street addresses to the map application to show the location.
- **all :** Enables all types of linking.

Turning on the autoLink feature relies on the detection of the various types within the Android SDK. In some cases, the linking might not be correct or might be misleading.



This is a test. My blog is at http://androidbook.blogspot.com
Email me at test@gmail.com
Call me at (800) 555-1212
Find me at my favorite coffee joint 1456 White Mountain Hwy, North Conway, NH 03860
—
This is a test. My blog is at http://androidbook.blogspot.com
Email me at test@gmail.com
Call me at (800) 555-1212
Find me at my favorite coffee joint 1456 White Mountain Hwy, North Conway, NH 03860
—
This is a test. My blog is at http://androidbook.blogspot.com
Email me at test@gmail.com
Call me at (800) 555-1212
Find me at my favorite coffee joint 1456 White Mountain Hwy, North Conway, NH 03860
—

**Figure 8.1 Three TextViews : Simple, AutoLink All (not clickable), and AutoLink All (clickable).**

Here is an example that links email and web pages, which, in our opinion, are the most reliable and predictable :

```
<TextView
android:id="@+id/TextView02"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/autolink_test"
android:autoLink="web|email" />
```

There are two helper values for this attribute, as well.You can set it to none to make sure no type of data is linked. You can also set it to all to have all known types linked. Figure 8.1 illustrates what happens when you click on these links.The default for a TextView is not to link any types. If you want the user to see the various data types highlighted but you don't want the user to click on them, you can set the linksClickable attribute to false.

## 8.5 Retrieving Text Input Using EditText Controls :

The Android SDK provides a convenient control called EditText to handle text input from a user. The EditText class is derived from TextView. In fact, most of its functionality is contained within TextView but enabled when created as an EditText. The EditText object has a number of useful features enabled by default, many of which are shown in Figure 8.2.

First, though, let's see how to define an EditText control in an XML layout file :

```
<EditText
android:id="@+id/EditText01"
android:layout_height="wrap_content"
android:hint="type here"
android:lines="4"
android:layout_width="fill_parent" />
```

This layout code shows a basic EditText element. There are a couple of interesting things to note. First, the hint attribute puts some text in the edit box that goes away when the user starts entering text. Essentially, this gives a hint to the user as to what should go there. Next is the lines attribute, which defines how many lines tall the input box is. If this is not set, the entry field grows as the user enters text. However, setting a size allows the user to scroll within a fixed sized to edit the text.This also applies to the width of the entry. By default, the user can perform a long press to bring up a context menu.This provides to the user some basic copy, cut, and paste operations as well as the ability to change the input method and add a word to the user's dictionary of frequently used words (shown in Figure 8.4).You do not need to provide any additional code for this useful behavior to benefit your users.You can also highlight a portion of the text from code, too.A call to setSelection() does this, and a call to selectAll() highlights the entire text entry field.

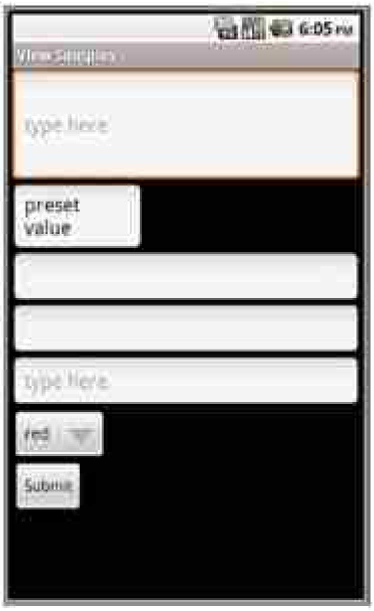


**Figure 8.2 : Various styles of EditText controls and Spinner and Button controls.**

The EditText object is essentially an editable TextView. This means that you can read text from it in the same way as you did with TextView: by using the getText() method.

You can also set initial text to draw in the text entry area using the setText() method.

This is useful when a user edits a form that already has data. Finally, you can set the editable attribute to false, so the user cannot edit the text in the field but can still copy text out of it using a long press.

**Helping the User with Auto Completion**

In addition to providing a basic text editor with the EditText control, the Android SDK also provides a way to help the user with entering commonly used data into forms.This functionality is provided through the auto–complete feature.

There are two forms of auto–complete. One is the more standard style of filling in the entire text entry based on what the user types. If the user begins typing a string that matches a word in a developer–provided list, the user can choose to complete the word with just a tap. This is done through the AutoCompleteTextView control (see Figure 8.3, left).The second method allows the user to enter a list of items, each of which has autocomplete functionality .These items must be separated in some way by providing a Tokenizer to the MultiAutoCompleteTextView object that handles this method. A common Tokenizer implementation is provided for comma–separated lists and is used by specifying the MultiAutoCompleteTextView.CommaTokenizer object. This can be helpful for lists of specifying common tags and the like.

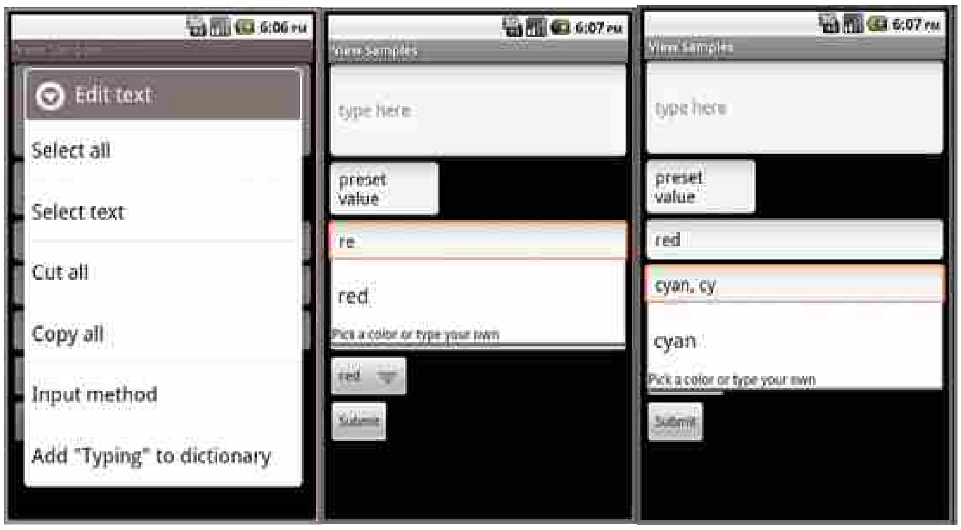


**Figure 8.3 : (left) A long press on EditText controls typically launches a Context menu for Select, Cut, and Paste. Using AutoCompleteTextView (center) and MultiAutoCompleteTextView (right).**

Both of the auto–complete text editors use an adapter to get the list of text that they use to provide completions to the user.This example shows how to provide an AutoCompleteTextView for the user that can help them type some of the basic colors from an array in the code :

```
final String[] COLORS = {
"red", "green", "orange", "blue", "purple",
"black", "yellow", "cyan", "magenta" };
ArrayAdapter<String> adapter =
new ArrayAdapter<String>(this,
android.R.layout.simple_dropdown_item_1line,
COLORS);
```

```
AutoCompleteTextView text = (AutoCompleteTextView)
findViewById(R.id.AutoCompleteTextView01);
text.setAdapter(adapter);
```

In this example, when the user starts typing in the field, if he starts with one of the letters in the COLORS array, a drop–down list shows all the available completions. Note that this does not limit what the user can enter. The user is still free to enter any text (such as puce). The adapter controls the look of the drop–down list. In this case,we use a built–in layout made for such things. Here is the layout resource definition for this

```
AutoCompleteTextView control :
<AutoCompleteTextView
android:id="@+id/AutoCompleteTextView01"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:completionHint=
"Pick a color or type your own"
android:completionThreshold="1" />
```

There are a couple more things to notice here. First, you can choose when the completion drop–down list shows by filling in a value for the completionThreshold attribute. In this case,we set it to a single character, so it displays immediately if there is a match.The default value is two characters of typing before it displays auto–completion options. Second, you can set some text in the completionHint attribute.This displays at the bottom of the drop–down list to help users. Finally, the drop–down list for completions is sized to the TextView. This means that it should be wide enough to show the completions and the text for the completionHint attribute.

The MultiAutoCompleteTextView is essentially the same as the regular auto–complete, except that you must assign a Tokenizer to it so that the control knows where each autocompletion should begin.The following is an example that uses the same adapter as the previous example but includes a Tokenizer for a list of user color responses, each separated by a comma :

```
MultiAutoCompleteTextView mtext =
(MultiAutoCompleteTextView)
  findViewById(R.id.MultiAutoCompleteTextView01);
mtext.setAdapter(adapter);
mtext.setTokenize
  (new MultiAutoCompleteTextView.CommaTokenizer());
```

As you can see, the only change is setting the Tokenizer. Here we use the built–in comma Tokenizer provided by the Android SDK. In this case, whenever a user chooses a color from the list, the name of the color is completed, and a comma is automatically added so that the user can immediately start typing in the next color.As before, this does not limit what the user can enter. If the user enters "maroon" and places a comma after it, the auto–completion starts again as the user types another color, regardless of the fact that it didn't help the user type in the color maroon.You can create your own

Tokenizer by implementing the MultiAutoCompleteTextView.Tokenizer interface. You can do this if you'd prefer entries separated by a semicolon or some other more complex separators.

**Constraining User Input with Input Filters**

There are often times when you don't want the user to type just anything. Validating input after the user has entered something is one way to do this. However, a better way to avoid wasting the user's time is to filter the input.The EditText control provides a way to set an InputFilter that does only this.

The Android SDK provides some InputFilter objects for use.There are InputFilter objects that enforce such rules as allowing only uppercase text and limiting the length of the text entered.You can create custom filters by implementing the InputFilter interface, which contains the single method called filter(). Here is an example of an EditText control with two built–in filters that might be appropriate for a two–letter state abbreviation :

```
final EditText text_filtered =
(EditText) findViewById(R.id.input_filtered);
text_filtered.setFilters(new InputFilter[] {
new InputFilter.AllCaps(),
new InputFilter.LengthFilter(2)
});
```

The setFilters() method call takes an array of InputFilter objects. This is useful for combining multiple filters, as shown. In this case, we convert all input to uppercase. Additionally, we set the maximum length to two characters long. The EditText control looks the same as any other, but if you try to type lowercase, the text is converted to uppercase, and the string is limited to two characters.This does not mean that all possible inputs are valid, but it does help users to not concern themselves with making the input too long or bother with the case of the input. This also helps your application by guaranteeing that any text from this input is a length of two. It does not constrain the input to only letters, though. Input filters can also be defined in XML.

## 8.6 Giving Users Input Choices Using Spinner Controls :

Sometimes you want to limit the choices available for users to type. For instance, if users are going to enter the name of a state, you might as well limit them to only the valid states because this is a known set. Although you could do this by letting them type something and then blocking invalid entries, you can also provide similar functionality with a Spinner control. As with the auto–complete method, the possible choices for a spinner can come from an Adapter. You can also set the available choices in the layout definition by using the entries attribute with an array resource (specifically a string–array that is referenced as something such as @array/state–list).The Spinner control isn't actually an EditText, although it is frequently used in a similar fashion. Here is an example of the XML layout definition for a Spinner control for choosing a color :

```
<Spinner
android:id="@+id/Spinner01"
android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
android:entries="@array/colors"
android:prompt="@string/spin_prompt" />
```

This places a Spinner control on the screen (see Figure 1.4).When the user selects it, a pop–up shows the prompt text followed by a list of the possible choices.This list allows only a single item to be selected at a time, and when one is selected, the pop–up goes away.

There are a couple of things to notice here. First, the entries attribute is set to the values that shows by assigning it to an array resource, referred to here as @array/colors.

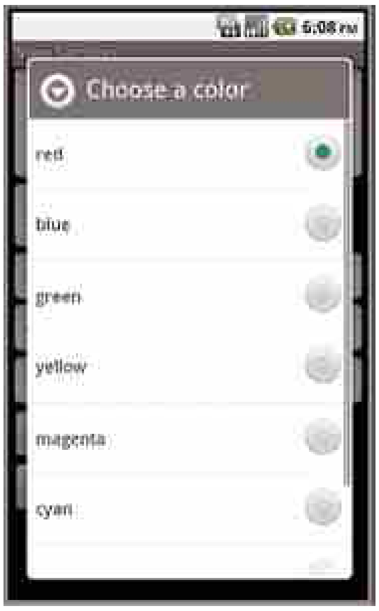


**Figure 8.4 : Filtering choices with a spinner control.**

Second, the prompt attribute is defined to a string resource. Unlike some other string attributes, this one is required to be a string resource. The prompt displays when the popup comes up and can be used to tell the user what kinds of values that can be selected from.

Because the Spinner control is not a TextView, but a list of TextView objects, you can't directly request the selected text from it. Instead, you have to retrieve the selected View and extract the text directly :

```
final Spinner spin = (Spinner) findViewById(R.id.Spinner01);
TextView text_sel = (TextView)spin. getSelectedView();
String selected_text = text_sel.getText();
```

As it turns out, you can request the currently selected View object, which happens to be a TextView in this case.This enables us to retrieve the text and use it directly. Alternatively, we could have called the getSelectedItem() or getSelectedItemId() methods to deal with other forms of selection.

## 8.7 Using Buttons, Check Boxes, and Radio Groups :

Another common user interface element is the button. In this section, you learn about different kinds of buttons provided by the Android SDK. These include the basic Button, ToggleButton, CheckBox, and RadioButton.You can find examples of each button type in Figure 8.5.

A basic Button is often used to perform some sort of action, such as submitting a form or confirming a selection.A basic Button control can contain a text or image label.

A CheckBox is a button with two states–checked or unchecked.You often use CheckBox controls to turn a feature on or off or to pick multiple items from a list.

A ToggleButton is similar to a CheckBox, but you use it to visually show the state.The default behavior of a toggle is like that of a power on/off button.

A RadioButton provides selection of an item. Grouping RadioButton controls together in a container called a RadioGroup enables the developer to enforce that only one RadioButton is selected at a time.

**8.7.1 Using Basic Buttons :**

The android.widget.Button class provides a basic button implementation in the Android SDK. Within the XML layout resources, buttons are specified using the Button element. The primary attribute for a basic button is the text field. This is the label that appears on the middle of the button's face. You often use basic Button controls for buttons with text such as "Ok,""Cancel," or "Submit."

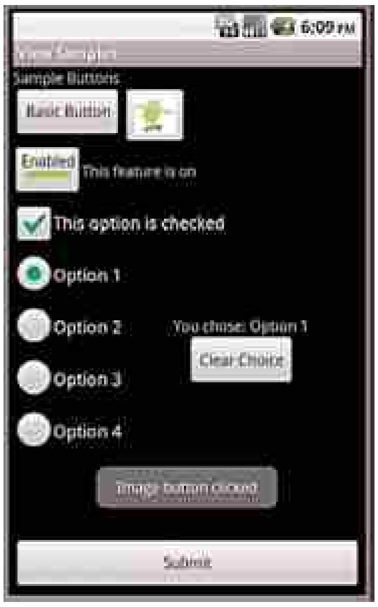


**Figure 8.5 : Various types of button controls.**

The following XML layout resource file shows a typical Button control definition :

```
<Button
android:id="@+id/basic_button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Basic Button" />
```

A button won't do anything, other than animate, without some code to handle the click event. Here is an example of some code that handles a click for a basic button and displays a Toast message on the screen :

```
setContentView(R.layout.buttons);
final Button basic_button =
(Button) findViewById(R.id.basic_button);
basic_button.setOnClickListener
(new View.OnClickListener() {
public void onClick(View v) {
Toast.makeText(ButtonsActivity.this,
"Button clicked", Toast.LENGTH_SHORT).show();
}
});
```

To handle the click event for when a button is pressed, we first get a reference to the Button by its resource identifier. Next, the setOnClickListener() method is called. It requires a valid instance of the class View.OnClickListener.A simple way to provide this is to define the instance right in the method call.This requires implementing the onClick() method.Within the onClick() method, you are free to carry out whatever actions you need. Here,we simply display a message to the users telling them that the button was, in fact, clicked.

A button with its primary label as an image is an ImageButton. An ImageButton is, for most purposes, almost exactly like a basic button. Click actions are handled in the same way. The primary difference is that you can set its src attribute to be an image. Here is an example of an ImageButton definition in an XML layout resource file :

```
<ImageButton
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/image_button"
android:src="@drawable/droid" />
```

**8.7.2 Using Check Boxes and Toggle Buttons :**

The check box button is often used in lists of items where the user can select multiple items. The Android check box contains a text attribute that appears to the side of the check box. This is used in a similar way to the label of a basic button. In fact, it's basically a TextView next to the button.

Here's an XML layout resource definition for a CheckBox control :

```
<CheckBox
android:id="@+id/checkbox"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
    android:text="Check me?" />
```

The following example shows how to check for the state of the button programmatically and change the text label to reflect the change :

```
final CheckBox check_button = (CheckBox)
     findViewById(R.id.checkbox);
check_button.setOnClickListener
(new View.OnClickListener() {
public void onClick (View v) {
TextView tv = (TextView)findViewById(R.id.checkbox);
tv.setText(check_button.isChecked() ?
"This option is checked" :
"This option is not checked");
}
     });
```

This is similar to the basic button. A check box automatically shows the check as enabled or disabled. This enables us to deal with behavior in our application rather than worrying about how the button should behave. The layout shows that the text starts out one way but, after the user presses the button, the text changes to one of two different things depending on the checked state. As the code shows, the CheckBox is also a TextView.

A Toggle Button is similar to a check box in behavior but is usually used to show or alter the on or off state of something. Like the CheckBox, it has a state (checked or not). Also like the check box, the act of changing what displays on the button is handled for us.

Unlike the CheckBox, it does not show text next to it. Instead, it has two text fields. The first attribute is textOn, which is the text that displays on the button when its checked state is on. The second attribute is textOff, which is the text that displays on the button when its checked state is off. The default text for these is "ON" and "OFF," respectively.

The following layout code shows a definition for a toggle button that shows "Enabled" or "Disabled" based on the state of the button :

```
<ToggleButton
android:id="@+id/toggle_button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Toggle"
android:textOff="Disabled"
     android:textOn="Enabled" />
```

### 8.7.3 Using RadioGroups and RadioButtons :

You often use radio buttons when a user should be allowed to only select one item from a small group of items. For instance, a question asking for gender can give three options : male, female, and unspecified. Only one of these options should be checked at a time. The RadioButton objects are similar to CheckBox objects. They have a text label next to them, set via the text attribute, and they have a state (checked or unchecked). However, you can group RadioButton objects inside a RadioGroup that handles enforcing their combined states so that only one RadioButton can be checked at a time. If the user selects a

RadioButton that is already checked, it does not become unchecked. However, you can provide the user with an action to clear the state of the entire RadioGroup so that none of the buttons are checked.

Here we have an XML layout resource with a RadioGroup containing four RadioButton objects (shown in Figure 8.7, toward the bottom of the screen). The RadioButton objects have text labels, "Option 1," and so on. The XML layout resource definition is shown here :

```
<RadioGroup
android:id="@+id/RadioGroup01"
android:layout_width="wrap_content"
android:layout_height="wrap_content">
<RadioButton
android:id="@+id/RadioButton01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Option 1"></RadioButton>
<RadioButton
android:id="@+id/RadioButton02"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Option 2"></RadioButton>
<RadioButton
android:id="@+id/RadioButton03"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Option 3"></RadioButton>
<RadioButton
android:id="@+id/RadioButton04"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Option 4"></RadioButton>
</RadioGroup>
```

You handle actions on these RadioButton objects through the RadioGroup object. The following example shows registering for clicks on the RadioButton objects within the RadioGroup :

```
final RadioGroup group =
(RadioGroup)findViewById(R.id.RadioGroup01);
final TextView tv = (TextView)
findViewById(R.id.TextView01);
group.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {
```

```
public void onCheckedChanged
( RadioGroup group, int checkedId) {
if (checkedId != -1) {
RadioButton rb = (RadioButton)
findViewById(checkedId);
if (rb != null) {
tv.setText("You chose: " + rb.getText());
}
} else {
tv.setText("Choose 1");
}
}
});
```

As this layout example demonstrates, there is nothing special that you need to do to make the RadioGroup and internal RadioButton objects work properly. The preceding code illustrates how to register to receive a notification whenever the RadioButton selection changes.

The code demonstrates that the notification contains the resource identifier for the specific RadioButton that the user chose, as assigned in the layout file.To do something interesting with this, you need to provide a mapping between this resource identifier (or the text label) to the corresponding functionality in your code. In the example,we query for the button that was selected, get its text, and assign its text to another TextView control that we have on the screen.

As mentioned, the entire RadioGroup can be cleared so that none of the RadioButton objects are selected.The following example demonstrates how to do this in response to a button click outside of the RadioGroup :

```
final Button clear_choice =
(Button) findViewById(R.id.Button01);
clear_choice.setOnClickListener
(new View.OnClickListener() {
public void onClick(View v) {
RadioGroup group = (RadioGroup)
findViewById(R.id.RadioGroup01);
if (group != null) {
group.clearCheck();
}
}
}
```

The action of calling the clearCheck() method triggers a call to the onCheckedChangedListener() callback method.This is why we have to make sure that the resource identifier we received is valid. Right after a call to the clearCheck() method, it is not a valid identifier but instead is set to the value –1 to indicate that no RadioButton is currently checked.

## 8.8 Getting Dates and Times from Users :

The Android SDK provides a couple controls for getting date and time input from the user. The first is the DatePicker control (Figure 8.6, top). It can be used to get a month, day, and year from the user.

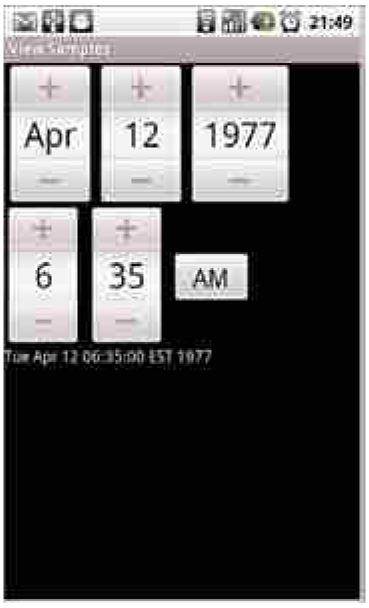


**Figure 8.6 : Date and time controls.**

The basic XML layout resource definition for a DatePicker follows :

```
<DatePicker
android:id="@+id/DatePicker01"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

As you can see from this example, there aren't any attributes specific to the DatePicker control. As with many of the other controls, your code can register to receive a method call when the date changes.You do this by implementing the onDateChanged() method.

However, this isn't done the usual way.

```
final  DatePicker  date  =
     (DatePicker)findViewById(R.id.DatePicker01);
date.init(date.getYear(),  date.getMonth(),
     date.getDayOfMonth(),
new  DatePicker.OnDateChangedListener()  {
public void onDateChanged(DatePicker view, int year,
int  monthOfYear,  int  dayOfMonth)  {
Date  dt  =  new  Date(year-1900,  monthOfYear,
     dayOfMonth,  time.getCurrentHour(),
time.getCurrentMinute());
text.setText(dt.toString());
}
    });
```

The preceding code sets the DatePicker.OnDateChangedListener by a call to the DatePicker.init() method. A DatePicker control is initialized with the current date. A TextView is set with the date value that the user entered into the DatePicker control. The value of 1900 is subtracted from the year parameter to make it compatible with the java.util.Date class.

A TimePicker control (also shown in Figure 8.8, bottom) is similar to the DatePicker control. It also doesn't have any unique attributes. However, to register for a method call when the values change, you call the more traditional method of

```
TimePicker.setOnTimeChangedListener().
time.setOnTimeChangedListener(new
    TimePicker.OnTimeChangedListener() {
public void onTimeChanged(TimePicker view,
int hourOfDay, int minute) {
Date dt = new Date(date.getYear()-1900,
    date.getMonth(),
date.getDayOfMonth(), hourOfDay, minute);
text.setText(dt.toString());
}
});
```

As in the previous example, this code also sets a TextView to a string displaying the time value that the user entered.When you use the DatePicker control and the TimePicker control together, the user can set a full date and time.

## 8.9 Using Indicators to Display Data to Users :

The Android SDK provides a number of controls that can be used to visually show some form of information to the user. These indicator controls include progress bars, clocks, and other similar controls.

### 8.9.1 Indicating Progress with ProgressBar :

Applications commonly perform actions that can take a while. A good practice during this time is to show the user some sort of progress indicator that informs the user that the application is off "doing something." Applications can also show how far a user is through some operation, such as a playing a song or watching a video. The Android SDK provides several types of progress bars.

The standard progress bar is a circular indicator that only animates. It does not show how complete an action is. It can, however, show that something is taking place. This is useful when an action is indeterminate in length.There are three sizes of this type of progress indicator (See Figure 8.7)

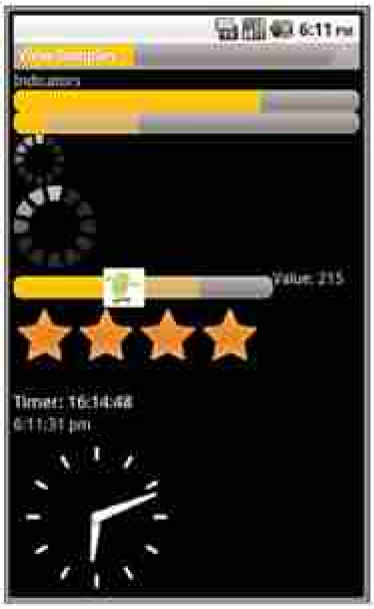


**Figure 8.7 : Various types of progress and rating indicators.**

The second type is a horizontal progress bar that shows the completeness of an action. (For example, you can see how much of a file is downloading.) This horizontal progress bar can also have a secondary progress indicator on it.This can be used, for instance, to show the completion of a downloading media file while that file plays.

This is an XML layout resource definition for a basic indeterminate progress bar :

```
<ProgressBar
android:id="@+id/progress_bar"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

The default style is for a medium–size circular progress indicator; not a "bar" at all. The other two styles for indeterminate progress bar are progressBarStyleLarge and progressBarStyleSmall. This style animates automatically. The next sample shows the layout definition for a horizontal progress indicator :

```
<ProgressBar
android:id="@+id/progress_bar"
style="?android:attr/progressBarStyleHorizontal"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
    android:max="100" />
```

We have also set the attribute for max in this sample to 100. This can help mimic a percentage progress bar. That is, setting the progress to 75 shows the indicator at 75 percent complete.

We can set the indicator progress status programmatically as follows :

```
mProgress =
(ProgressBar) findViewById(R.id.progress_bar);
mProgress.setProgress(75);
```

You can also put these progress bars in your application's title bar (as shown in Figure 8.9).

This can save screen real estate. This can also make it easy to turn on and off an indeterminate progress indicator without changing the look of the screen. Indeterminate progress indicators are commonly used to display progress on pages where items need to be loaded before the page can finish drawing. This is often employed on web browser screens. The following code demonstrates how to place this type of indeterminate progress indicator on your Activity screen :

```
requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
requestWindowFeature(Window.FEATURE_PROGRESS);
setContentView(R.layout.indicators);
setProgressBarIndeterminateVisibility(true);
setProgressBarVisibility(true);
setProgress(5000);
```

To use the indeterminate indicator on your Activity objects title bar, you need to request the feature Window. FEATURE_INDETERMINATE_ PROGRESS, as previously shown. This shows a small circular indicator in the right side of the title bar. For a horizontal progress bar style that shows behind the title, you need to enable the Window.FEATURE_PROGRESS.

These features must be enabled before your application calls the setContentView() method, as shown in the preceding example.

You need to know about a couple of important default behaviors. First, the indicators are visible by default. Calling the visibility methods shown in the preceding example can set their visibility on or off. Second, the horizontal progress bar defaults to a maximum progress value of 10,000. In the preceding example,we set it to 5,000, which is equivalent to 50 percent.When the value reaches the maximum value, the indicators fade away so that they aren't visible.This happens for both indicators.

### 8.9.2 Adjusting Progress with SeekBar :

You have seen how to display progress to the user. What if, however, you want to give the user some ability to move the indicator, for example, to set the current cursor position in a playing media file or to tweak a volume setting ? You accomplish this by using the SeekBar control provided by the Android SDK. It's like the regular horizontal progress bar, but includes a thumb, or selector, that can be dragged by the user. A default thumb selector is provided, but you can use any drawable item as a thumb. In Figure 8.7 (center),we replaced the default thumb with a little Android graphic.

Here we have an example of an XML layout resource definition for a simple SeekBar :

```
<SeekBar
android:id="@+id/seekbar1"
android:layout_height="wrap_content"
android:layout_width="240px"
android:max="500" />
```

With this sample SeekBar, the user can drag the thumb to any value between 0 and 500. Although this is shown visually, it might be useful to show the user what exact value the user is selecting.To do this, you can provide an implementation of the onProgressChanged() method, as shown here :

```
SeekBar seek =
(SeekBar) findViewById(R.id.seekbar1);
seek.setOnSeekBarChangeListener(
new SeekBar.OnSeekBarChangeListener() {
public void onProgressChanged(
SeekBar seekBar, int progress,boolean fromTouch) {
((TextView)findViewById(R.id.seek_text))
.setText("Value: "+progress);
seekBar.setSecondaryProgress(
(progress+seekBar.getMax())/2);
}
});
```

There are two interesting things to notice in this example. The first is that the fromTouch parameter tells the code if the change came from the user input or if, instead, it came from a programmatic change as demonstrated with the regular ProgressBar controls. The second interesting thing is that the SeekBar still enables you to set a secondary progress value. In this example, we set the secondary indicator to be halfway between the user's selected value and the maximum value of the progress bar.You might use this feature to show the progress of a video and the buffer stream.

**8.9.3 Displaying Rating Data with RatingBar :**

Although the SeekBar is useful for allowing a user to set a value, such as the volume, the RatingBar has a more specific purpose : showing ratings or getting a rating from a user. By default, this progress bar uses the star paradigm with five stars by default. A user can drag across this horizontal to set a rating. A program can set the value, as well. However, the secondary indicator cannot be used because it is used internally by this particular control.

Here's an example of an XML layout resource definition for a RatingBar with four stars :

```
<RatingBar
android:id="@+id/ratebar1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:numStars="4"
    android:stepSize="0.25" />
```

This layout definition for a RatingBar demonstrates setting both the number of stars and the increment between each rating value. Here, users can choose any rating value between 0 and 4.0, but only in increments of 0.25, the stepSize value. For instance, users can set a value of 2.25. This is visualized to the users, by default, with the stars partially filled. Figure 8.9 (center) illustrates how the RatingBar behaves.

Although the value is indicated to the user visually, you might still want to show a numeric representation of this value to the user. You can do this by implementing the onRatingChanged() method of the RatingBar.OnRatingBar ChangeListener class.

```
RatingBar rate =
(RatingBar) findViewById(R.id.ratebar1);
rate.setOnRatingBarChangeListener(new
RatingBar.OnRatingBarChangeListener() {
public void onRatingChanged(RatingBar ratingBar,
float rating, boolean fromTouch) {
((TextView)findViewById(R.id.rating_text))
.setText("Rating: "+ rating);
}
});
```

The preceding example shows how to register the listener. When the user selects a rating using the control, a TextView is set to the numeric rating the user entered. One interesting thing to note is that, unlike the SeekBar, the implementation of the onRatingChange() method is called after the change is complete, usually when the user lifts a finger.That is, while the user is dragging across the stars to make a rating, this method isn't called. It is called when the user stops pressing the control.

**8.9.4 Showing Time Passage with the Chronometer :**

Sometimes you want to show time passing instead of incremental progress. In this case, you can use the Chronometer control as a timer (see Figure 8.9, bottom). This might be useful if it's the user who is taking time doing some task or in a game where some action needs to be timed.The Chronometer control can be formatted with text, as shown in this XML layout resource definition :

```
<Chronometer
android:id="@+id/Chronometer01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:format="Timer: %s" />
```

You can use the Chronometer object's format attribute to put text around the time that displays. A Chronometer won't show the passage of time until its start() method is called. To stop it, simply call its stop() method. Finally, you can change the time from which the timer is counting. That is, you can set it to count from a particular time in the past instead of from the time it's started. You call the setBase() method to do this. In this next example code, the timer is retrieved from the View by its resource identifier.

We then check its base value and set it to 0. Finally,we start the timer counting up from there.

```
final Chronometer timer =
(Chronometer)findViewById(R.id.Chronometer01);
long base = timer.getBase();
```

```
Log.d(ViewsMenu.debugTag, "base = "+ base);
timer.setBase(0);
timer.start();
```

### 8.9.5 Displaying the Time :

Displaying the time in an application is often not necessary because Android devices have a status bar to display the current time.However, there are two clock controls available to display this information: the DigitalClock and AnalogClock controls.

**Using the DigitalClock**

The DigitalClock control (Figure 8.9, bottom) is a compact text display of the current time in standard numeric format based on the users' settings. It is a TextView, so anything you can do with a TextView you can do with this control, except change its text.You can change the color and style of the text, for example.

By default, the DigitalClock control shows the seconds and automatically updates as each second ticks by. Here is an example of an XML layout resource definition for a DigitalClock control :

```
<DigitalClock
android:id="@+id/DigitalClock01"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

**Using the AnalogClock**

The AnalogClock control (Figure 8.9, bottom) is a dial–based clock with a basic clock face with two hands, one for the minute and one for the hour. It updates automatically as each minute passes.The image of the clock scales appropriately with the size of its View.

Here is an example of an XML layout resource definition for an AnalogClock control :

```
<AnalogClock
android:id="@+id/AnalogClock01"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

The AnalogClock control's clock face is simple. However you can set its minute and hour hands. You can also set the clock face to specific drawable resources, if you want to jazz it up. Neither of these clock controls accepts a different time or a static time to display. They can show only the current time in the current time zone of the device, so they are not particularly useful.

## 8.10 Providing Users with Options and Context Menus :

You need to be aware of two special application menus for use within your Android applications : the options menu and the context menu.

### 8.10.1 Enabling the Options Menu :

The Android SDK provides a method for users to bring up a menu by pressing the menu key from within the application (see Figure 8.8). You can use options menus within your application to bring up help, to navigate, to provide additional controls, or to configure options. The OptionsMenu control can contain icons, submenus, and keyboard shortcuts.

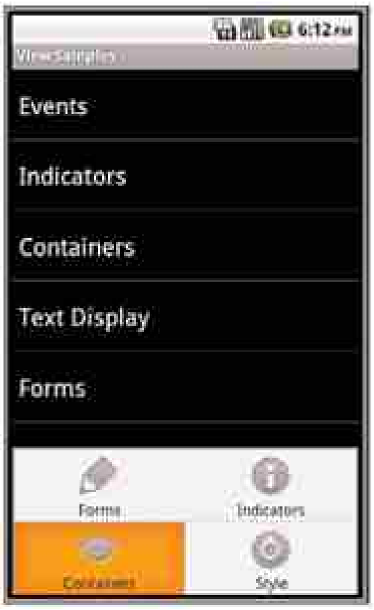


**Figure 8.8 : An Option Menu**

For an options menu to show when a user presses the Menu button on their device, you need to override the implementation of onCreateOptionsMenu() in your Activity.

Here is a sample implementation that gives the user three menu items to choose from :

```
public boolean onCreateOptionsMenu
(android.view.Menu menu) {
super.onCreateOptionsMenu(menu);
menu.add("Forms")
.setIcon(android.R.drawable.ic_menu_edit)
.setIntent(new Intent(this, FormsActivity.class));
menu.add("Indicators")
.setIntent(new Intent(this,
IndicatorsActivity.class))
.setIcon(android.R.drawable.ic_menu_info_details);
menu.add("Containers")
.setIcon(android.R.drawable.ic_menu_view)
.setIntent(new Intent(this,
ContainersActivity.class));
return true;
    }
```

To handle the event when a menu option item is selected, we also implement the onOptionsItemSelected() method, as shown here :

```
public boolean onOptionsItemSelected(MenuItem item) {
if (item.getItemId() == light_id) {
```

```
item.setChecked(true);
isLight = true;
return true;
} else if (item.getItemId() == dark_id) {
item.setChecked(true);
isLight = false;
return true;
}
return super.onOptionsItemSelected(item);
}
```

### 8.10.2 Enabling the ContextMenu :

The ContextMenu is a subtype of Menu that you can configure to display when a long press is performed on a View.As the name implies, the ContextMenu provides for contextual menus to display to the user for performing additional actions on selected items.

ContextMenu objects are slightly more complex than OptionsMenu objects. You need to implement the onCreateContextMenu() method of your Activity for one to display.

However, before that is called, you must call the registerForContextMenu() method and pass in the View for which you want to have a context menu.This means each View on your screen can have a different context menu, which is appropriate as the menus are designed to be highly contextual.

Here we have an example of a Chronometer timer, which responds to a long click with a context menu :

registerForContextMenu(timer);

After the call to the registerForContextMenu() method has been executed, the user can then long click on the View to open the context menu. Each time this happens, your Activity gets a call to the onCreateContextMenu() method, and your code creates the menu each time the user performs the long click.

The following is an example of a context menu for the Chronometer control, as previously used :

```
public void onCreateContextMenu(
ContextMenu menu, View v, ContextMenuInfo menuInfo) {
super.onCreateContextMenu(menu, v, menuInfo);
if (v.getId() == R.id.Chronometer01) {
getMenuInflater().inflate (R.menu.timer_context,
    menu);
menu.setHeaderIcon(android.R.drawable.ic_media_play)
.setHeaderTitle("Timer controls");
}
}
```

Recall that any View control can register to trigger a call to the onCreateContextMenu() method when the user performs a long press.That means we have to check which View control it was for which the user tried to get a context menu. Next, we inflate the appropriate menu from a menu resource that we defined with XML. Because we can't define header information in the menu resource file,we set a stock Android SDK resource to it and add a title. Here is the menu resource that is inflated :

```
<menu
xmlns:android=
"http://schemas.android.com/apk/res/android">
<item
android:id="@+id/start_timer"
android:title="Start" />
<item
android:id="@+id/stop_timer"
android:title="Stop" />
<item
android:id="@+id/reset_timer"
android:title="Reset" />
</menu>
```

Now we need to handle the ContextMenu clicks by implementing the onContextItemSelected() method in our Activity. Here's an example :

```
public boolean onContextItemSelected(MenuItem item) {
super.onContextItemSelected(item);
boolean result = false;
Chronometer timer =
(Chronometer)findViewById(R.id.Chronometer01);
switch (item.getItemId()){
case R.id.stop_timer:
timer.stop();
result = true;
break;
case R.id.start_timer:
timer.start();
result = true;
break;
case R.id.reset_timer:
timer.setBase(SystemClock.elapsedRealtime());
result = true;
break;
}
return  result;
}
```

## 8.11 Handling User Events :

You've seen how to do basic event handling in some of the previous control examples. For instance, you know how to handle when a user clicks on a button.There are a number of other events generated by various actions the user might take.This section briefly introduces you to some of these events. First, though,we need to talk about the input states within Android.

### 8.11.1 Listening for Touch Mode Changes :

The Android screen can be in one of two states. The state determines how the focus on View controls is handled.When touch mode is on, typically only objects such as EditText get focus when selected. Other objects, because they can be selected directly by the user tapping on the screen,won't take focus but instead trigger their action, if any. When not in touch mode, however, the user can change focus between even more object types. These include buttons and other views that normally need only a click to trigger their action. In this case, the user uses the arrow keys, trackball, or wheel to navigate between items and select them with the Enter or select keys.

Knowing what mode the screen is in is useful if you want to handle certain events. If, for instance, your application relies on the focus or lack of focus on a particular control, your application might need to know if the phone is in touch mode because the focus behavior is likely different.

Your application can register to find out when the touch mode changes by using the addOnTouchModeChangeListener() method within the android.view.ViewTreeObserver class. Your application needs to implement the ViewTreeObserver.OnTouchModeChangeListener class to listen for these events.

Here is a sample implementation:

```
View all = findViewById(R.id.events_screen);
ViewTreeObserver vto = all.getViewTreeObserver();
vto.addOnTouchModeChangeListener(
new ViewTreeObserver.OnTouchModeChangeListener() {
public void onTouchModeChanged(
boolean isInTouchMode) {
events.setText("Touch mode: " + isInTouchMode);
}
});
```

### 8.11.2 Listening for Events on the Entire Screen :

You saw in the last section how your application can watch for changes to the touch mode state for the screen using the ViewTreeObserver class. The ViewTreeObserver also provides three other events that can be watched for on a full screen or an entire View and all of its children. These are

- **PreDraw :** Get notified before the View and its children are drawn
- **GlobalLayout :** Get notified when the layout of the View and its children might change, including visibility changes
- **GlobalFocusChange :** Get notified when the focus within the View and its children changes

Your application might want to perform some actions before the screen is drawn. You can do this by calling the method addOnPreDrawListener() with an implementation of the ViewTreeObserver.OnPreDrawListener class interface.

Similarly, your application can find out when the layout or visibility of a View has changed. This might be useful if your application is dynamically changing the display contents of a view and you want to check to see if a View still fits on the screen. Your application needs to provide an implementation of the ViewTreeObserver.OnGlobalLayoutListener class interface to the addGlobalLayoutListener() method of the ViewTreeObserver object.

Finally, your application can register to find out when the focus changes between a View control and any of its child View controls. Your application might want to do this to monitor how a user moves about on the screen.When in touch mode, though, there might be fewer focus changes than when the touch mode is not set. In this case, your application needs to provide an implementation of the

```
ViewTreeObserver.OnGlobalFocusChangeListener class
     interface to the
addGlobalFocusChangeListener() method. Here is a sample
     implementation of this :
vto.addOnGlobalFocusChangeListener(new
ViewTreeObserver.OnGlobalFocusChangeListener() {
public void onGlobalFocusChanged(
View oldFocus, View newFocus) {
if (oldFocus != null && newFocus != null) {
events.setText("Focus \nfrom: " +
oldFocus.toString() + " \nto: " +
newFocus.toString());
}
}
     });
```

### 8.11.3 Listening for Long Clicks :

In a previous section discussing the ContextMenu control, you learned that you can add a context menu to a View that is activated when the user performs a long click on that view. A long click is typically when a user presses on the touch screen and holds his finger there until an action is performed. However, a long press event can also be triggered if the user navigates there with a non–touch method, such as via a keyboard or trackball, and then holds the Enter or Select key for a while. This action is also often called a press–andhold action.

Although the context menu is a great typical use case for the long–click event, you can listen for the long–click event and perform any action you want. However, this is the same event that triggers the context menu. If you've already added a context menu to a View, you might not want to listen for the long–click event as other actions or side effects might confuse the user or even prevent the context menu from showing. As always with good user interface design, try to be consistent for usability sake Your application can listen to the long–click event on any View.The following example demonstrates how to listen for a long–click event on a Button control :

```
Button long_press =
(Button)findViewById(R.id.long_press);
long_press.setOnLongClickListener(new
    View.OnLongClickListener() {
public boolean onLongClick(View v) {
events.setText("Long click: " + v.toString());
return true;
}
    });
```

### 8.11.4 Listening for Focus Changes :

We already discussed focus changes for listening for them on an entire screen. All View objects, though, can also trigger a call to listeners when their particular focus state changes. You do this by providing an implementation of the View.OnFocusChangeListener class to the setOnFocusChangeListener() method.The following is an example of how to listen for focus change events with an EditText control :

```
TextView focus =
(TextView)findViewById(R.id.text_focus_change);
focus.setOnFocusChangeListener(new
    View.OnFocusChangeListener() {
public void onFocusChange(View v, boolean hasFocus) {
if (hasFocus) {
if (mSaveText != null) {
((TextView)v).setText(mSaveText);
}
} else {
mSaveText = ((TextView)v).getText().toString();
((TextView)v).setText("");
}
    }
```

## 8.12 Working with Dialogs :

An Activity can use dialogs to organize information and react to user–driven events. For example, an activity might display a dialog informing the user of a problem or ask the user to confirm an action such as deleting a data record. Using dialogs for simple tasks helps keep the number of application activities manageable

### 8.12.1 Exploring the Different Types of Dialogs :

There are a number of different dialog types available within the Android SDK. Each has a special function that most users should be somewhat familiar with.The dialog types available include n Dialog : The basic class for all Dialog types. A basic Dialog is shown in the top left of Figure.

- **AlertDialog :** A Dialog with one, two, or three Button controls. An AlertDialog is shown in the top center of Figure.
- **CharacterPickerDialog :** A Dialog for choosing an accented character associated with a base character. A CharacterPickerDialog is shown in the top right of Figure.
- **DatePickerDialog :** A Dialog with a DatePicker control. A DatePickerDialog is shown in the bottom left of Figure.
- **ProgressDialog :** A Dialog with a determinate or indeterminate ProgressBar control. An indeterminate ProgressDialog is shown in the bottom center of Figure.
- **TimePickerDialog :** A Dialog with a TimePicker control. A TimePickerDialog is shown in the bottom right of Figure

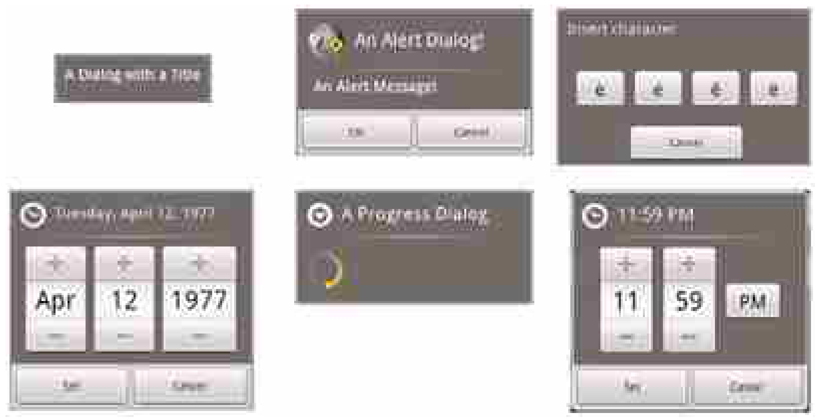


**Figure 8.9 : The different dialog types available in Android.**

### 8.12.2 Tracing the Lifecycle of a Dialog :

Each Dialog must be defined within the Activity in which it is used.A Dialog may be launched once, or used repeatedly. Understanding how an Activity manages the Dialog lifecycle is important to implementing a Dialog correctly. Let's look at the key methods that an Activity must use to manage a Dialog :

- The showDialog() method is used to display a Dialog.
- The dismissDialog() method is used to stop showing a Dialog.The Dialog is kept around in the Activity's Dialog pool. If the Dialog is shown again using showDialog(), the cached version is displayed once more.
- The removeDialog() method is used to remove a Dialog from the Activity objects Dialog pool. The Dialog is no longer kept around for future use. If you call showDialog() again, the Dialog must be re–created.

Adding the Dialog to an Activity involves several steps :

1. Define a unique identifier for the Dialog within the Activity.

2. Implement the onCreateDialog() method of the Activity to return a Dialog of the appropriate type, when supplied the unique identifier.
3. Implement the onPrepareDialog() method of the Activity to initialize the Dialog as appropriate.
4. Launch the Dialog using the showDialog() method with the unique identifier.

## 8.13 Working with Styles :

A style is a group of common View attribute values. You can apply the style to individual View controls. Styles can include such settings as the font to draw with or the color of text. The specific attributes depend on the View drawn. In essence, though, each style attribute can change the look and feel of the particular object drawn.

In the previous examples of this chapter, you have seen how XML layout resource files can contain many references to attributes that control the look of TextView objects. You can use a style to define your application's standard TextView attributes once and then reference to the style either in an XML layout file or programmatically from within Java.

In Chapter 6, we see how you can use one style to indicate mandatory form fields and another to indicate optional fields. Styles are typically defined within the resource file res/values/styles.xml. The XML file consists of a resources tag with any number of style tags, which contain an item tag for each attribute and its value that is applied with the style.

The following is an example with two different styles :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<style name="padded_small">
<item name="android:padding">2px</item>
<item name="android:textSize">8px</item>
</style>
<style name="padded_large">
<item name="android:padding">4px</item>
<item name="android:textSize">16px</item>
</style>
    </resources>
```

When applied, this style sets the padding to two pixels and the textSize to eight pixels.

The following is an example of how it is applied to a TextView from within a layout resource file :

```
<TextView
style="@style/padded_small"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Small Padded" />
```

Styles support inheritance; therefore, styles can also reference another style as a parent.

This way, they pick up the attributes of the parent style. The following is an example of how you might use this :

```
<style name="red_padded">
<item name="android:textColor">#F00</item>
<item name="android:padding">3px</item>
</style>
<style name="padded_normal" parent="red_padded">
<item name="android:textSize">12px</item>
</style>
<style name="padded_italics" parent="red_padded">
<item name="android:textSize">14px</item>
<item name="android:textStyle">italic</item>
</style>
```

Here you find two common attributes in a single style and a reference to them from the other two styles that have different attributes. You can reference any style as a parent style; however, you can set only one style as the style attribute of a View. Applying the padded_italics style that is already defined makes the text 14 pixels in size, italic, red, and padded. The following is an example of applying this style :

```
<TextView
style="@style/padded_italics"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Italic w/parent color" />
```

As you can see from this example, applying a style with a parent is no different than applying a regular style. In fact, a regular style can be used for applying to Views and used as a parent in a different style.

```
<style name="padded_xlarge">
<item name="android:padding">10px</item>
<item name="android:textSize">100px</item>
</style>
    <style name="green_glow" parent="padded_xlarge">
<item name="android:shadowColor">#0F0</item>
<item name="android:shadowDx">0</item>
<item name="android:shadowDy">0</item>
<item name="android:shadowRadius">10</item>
</style>
```

Here the padded_xlarge style is set as the parent for the green_glow style. All six attributes are then applied to any view that this style is set to.

## 8.14 Working with Themes :

A theme is a collection of one or more styles (as defined in the resources) but instead of applying the style to a specific control, the style is applied to all View objects within a specified Activity. Applying a theme to a set of View objects all at once simplifies making the user interface look consistent and can be a great way to define color schemes and other common control attribute settings.

An Android theme is essentially a style that is applied to an entire screen. You can specify the theme programmatically by calling the Activity method setTheme() with the style resource identifier. Each attribute of the style is applied to each View within that Activity, as applicable. Styles and attributes defined in the layout files explicitly override those in the theme.

For instance, consider the following style :

```
<style name="right">
<item name="android:gravity">right</item>
</style>
```

You can apply this as a theme to the whole screen, which causes any view displayed within that Activity to have its gravity attribute to be right–justified. Applying this theme is as simple as making the method call to the setTheme() method from within the Activity, as shown here :

```
setTheme(R.style.right);
```

You can also apply themes to specific Activity instances by specifying them as an attribute within the <activity> element in the AndroidManifest.xml file, as follows :

```
<activity android:name=".myactivityname"
android:label="@string/app_name"
android:theme="@style/myAppIsStyling">
```

Unlike applying a style in an XML layout file, multiple themes can be applied to a screen.

This gives you flexibility in defining style attributes in advance while applying different configurations of the attributes based on what might be displayed on the screen. This is demonstrated in the following code :

```
setTheme(R.style.right);
setTheme(R.style.green_glow);
setContentView(R.layout.style_samples);
```

In this example, both the right style and the green_glow style are applied as a theme to the entire screen.You can see the results of green glow and right–aligned gravity, applied to a variety of TextView controls on a screen, as shown in Figure 8.10. Finally, we set the layout to the Activity. You must do this after setting the themes. That is, you must apply all themes before calling the method setContentView() or the inflate() method so that the themes' attributes can take effect.

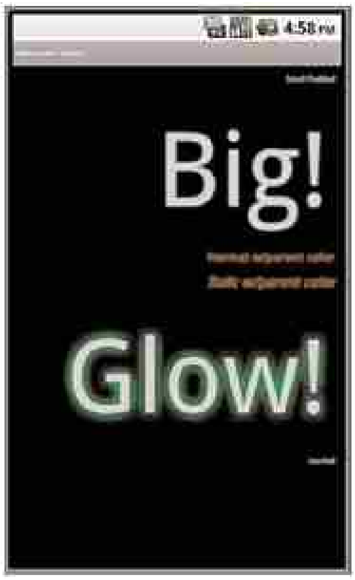


**Figure 8.10 : Packaging styles for glowing text, padding, and alignment into a theme.**

A combination of well–designed and thought–out themes and styles can make the look of your application consistent and easy to maintain. Android comes with a number of built–in themes that can be a good starting point. These include such themes as Theme_Black, Theme_Light, and Theme_NoTitleBar_Fullscreen, as defined in the android.R.style class. They are all variations on the system theme, Theme, which built–in apps use.

❑ **Check Your Progress :**

1. Which of the following method is used to handle what happens after clicking a button ?

   (A) onPause() (B) onRestart() (C) onStop() (D) onClick()

2. In which state the activity is, if it is not in focus, but still visible on the screen ?

   (A) pause state (B) start state (C) restart state (D) stop state

3. All layout classes are the subclasses of ________.

   (A) android.java (B) android.view.ViewGroup

   (C) android.layout (D) android.permission

4. Which of the following layout in android arranges its children into rows and columns ?

   (A) RelativeLayout (B) CardLayout

   (C) TableLayout (D) None of the Above

5. A theme is a collection of one or more ________.

   (A) services (B) layouts (C) fragments (D) styles

## 8.15 Let Us Sum Up :

In this unit we have learnt how to use interface & Android control, learn how to create screen elements with control and to understand the views and layouts.

## 8.16 Answers for Check Your Progress :

**1.** (D) **2.** (A) **3.** (B) **4.** (C) **5.** (D)

## 8.17 Glossary :

| | |
|---|---|
| **1.** | **SDK :** Software Development Kit |
| **2.** | **AVD :** Android Virtual Device |
| **3.** | **DDMS :** Dalvik Debug Monitor Server |
| **4.** | **ADB :** Android Debug Bridge |

## 8.18 Assignment :

1. Explain different Layout available in Android.
2. Explain different Common user interface elements available in Android.
3. What is menu? Which two types of menus available in Android ?
4. Explain Different Types of Dialogs with lifecycle in Android.

## 8.19 Activities :

1. Use the various android control and arrange them with proper layouts available in the android.

## 8.20 Case Study :

Identify the system and design the user interface with proper layout and look and feel.

## 8.21 Further Reading :

Lauren Darcey and Shane Conder, "Android Wireless Application Development", Pearson Education, 2nd ed. (2011).

Unit 09

# DESIGNING USER INTERFACES WITH LAYOUTS

## UNIT STRUCTURE

## 9.0 Learning Objectives :

- To learn how to use user interface in Android
- To learn how to create screen elements
- To understand the views and layouts

## 9.1 Introduction :

In this unit, we discuss how to design user interfaces for Android applications. Here we focus on the various layout controls you can use to organize screen elements in different ways. We also cover some of the more complex View objects we call container views. These are View objects that can contain other View objects and controls.

## 9.2 Creating User Interfaces in Android :

Application user interfaces can be simple or complex, involving many different screens or only a few. Layouts and user interface controls can be defined as application resources or created programmatically at runtime.

### 9.2.1 Creating Layouts Using XML Resources :

As discussed in previous unit, Android provides a simple way to create layout files in XML as resources provided in the */res/layout* project directory. This is the most common and convenient way to build Android user interfaces and is especially useful for defining static screen elements and control properties that you know in advance, and to set default attributes that you can modify programmatically at runtime.

You can configure almost any ViewGroup or View (or View subclass) attribute using the XML layout resource files. This method greatly simplifies the user interface design process, moving much of the static creation and layout of user interface controls, and basic definition of control attributes, to the XML, instead of littering the code. Developers reserve the ability to alter these layouts programmatically as necessary, but they can set all the defaults in the XML template.

You'll recognize the following as a simple layout file with a LinearLayout and a single TextView control. This is the default layout file provided with any new Android project in Eclipse, referred to as /res/layout/main.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" >
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/hello" />
</LinearLayout>
```

This block of XML shows a basic layout with a single TextView. The first line, which you might recognize from most XML files, is required. Because it's common across all the files, we do not show it in any other examples.

Next, we have the LinearLayout element. LinearLayout is a ViewGroup that shows each child View either in a single column or in a single row.When applied to a full screen, it merely means that each child View is drawn under the previous View if the orientation is set to vertical or to the right of the previous View if orientation is set to horizontal.

Finally, there is a single child View–in this case, a TextView. A TextView is a control, which is also a View. A TextView draws text on the screen. In this case, it draws the text defined in the "@string/hello" string resource.

Creating only an XML file, though,won't actually draw anything on the screen. A particular layout is usually associated with a particular Activity. In your default Android project, there is only one activity, which sets the main.xml layout by default. To associate the main.xml layout with the activity, use the method call setContentView() with the identifier of the main.xml layout. The ID of the layout matches the XML filename without the extension. In this case, the preceding example came from main.xml, so the identifier of this layout is simply main :

setContentView(R.layout.main);

### 9.2.2 Creating Layouts Programmatically :

You can create user interface components such as layouts at runtime programmatically, but for organization and maintainability, it's best that you leave this for the odd case rather than the norm.The main reason is because the creation of layouts programmatically is onerous and difficult to maintain, whereas the XML resource method is visual, more organized, and could be done by a separate designer with no Java skills.

The following example shows how to programmatically have an Activity instantiate a LinearLayout view and place two TextView objects within it. No resources whatsoever are used; actions are done at runtime instead.

```
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
TextView text1 = new TextView(this);
text1.setText("Hi there!");
TextView text2 = new TextView(this);
text2.setText("I'm second. I need to wrap.");
text2.setTextSize((float) 60);
LinearLayout ll = new LinearLayout(this);
ll.setOrientation(LinearLayout.VERTICAL);
ll.addView(text1);
ll.addView(text2);
setContentView(ll);
}
```

The onCreate() method is called when the Activity is created. The first thing this method does is some normal Activity housekeeping by calling the constructor for the base class.

Next, two TextView controls are instantiated. The Text property of each TextView is set using the setText() method.All TextView attributes, such as TextSize, are set by making method calls on the TextView object. These actions perform the same function that you have in the past by setting the properties Text and TextSize using the Eclipse layout resource designer, except these properties are set at runtime instead of defined in the layout files compiled into your application package.

To display the TextView objects appropriately, we need to encapsulate them within a container of some sort (a layout). In this case, we use a LinearLayout with the orientation set to VERTICAL so that the second TextView begins beneath the first, each aligned to the left of the screen.The two TextView controls are added to the LinearLayout in the order we want them to display.

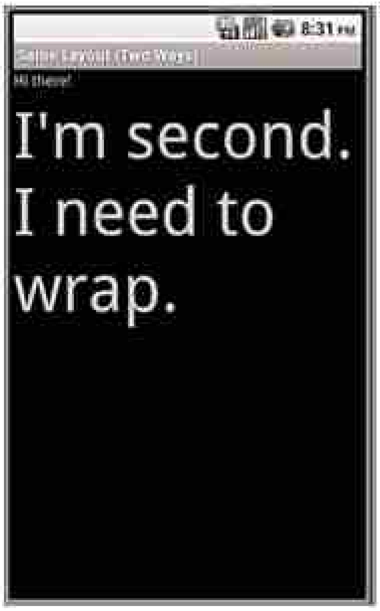
Finally, we call the setContentView() method, part of your Activity class, to draw the LinearLayout and its contents on the screen.

As you can see, the code can rapidly grow in size as you add more View controls and you need more attributes for each View. Here is that same layout, now in an XML layout file :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
>
<TextView
android:id="@+id/TextView1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Hi There!"
/>
<TextView
android:id="@+id/TextView2"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="60px"
android:text="I'm second. I need to wrap."
/>
</LinearLayout>
```

You might notice that this isn't a literal translation of the code example from the previous

section, although the output is identical, as shown in Figure 9.1.



```
Hi there!
I'm second.
I need to
wrap.
```

**Figure 9.1 : Two different methods to create a screen have the same result.**

First, in the XML layout files, layout_width and layout_height are required attributes.

Next, you see that each TextView object has a unique id property assigned so that it can be accessed programmatically at runtime. Finally, the textSize property needs to have its units defined. The XML attribute takes a dimension type (as described in Chapter 6) instead of a float.

The end result differs only slightly from the programmatic method. However, it's far easier to read and maintain. Now you need only one line of code to display this layout view. Again, if the layout resource is stored in the /res/layout/resource_based_layout.xml file, that is

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

## 9.3 Organizing Your User Interface :

In previous unit, we talk about how the class View is the building block for user interfaces in Android. All user interface controls, such as Button, Spinner, and EditText, derive from the View class.

Now we talk about a special kind of View called a ViewGroup. The classes derived from ViewGroup enable developers to display View objects on the screen in an organized fashion.

### 9.3.1 Understanding View versus ViewGroup :

Like other View objects, including the controls from previous unit, ViewGroup controls represent a rectangle of screen space. What makes ViewGroup different from your typical control is that ViewGroup objects contain other View objects. A View that contains other View objects is called a parent view. The parent View contains View objects called child views, or children.

You add child View objects to a ViewGroup programmatically using the method addView(). In XML, you add child objects to a ViewGroup by defining the child View control as a child node in the XML (within the parent XML element, as we've seen various times using the LinearLayout ViewGroup).

ViewGroup subclasses are broken down into two categories :

- Layout classes
- View container controls

### 9.3.2 Sub Topic :

Check Your Progress

## 9.4 Using Built–In Layout Classes :

We talked a lot about the LinearLayout layout, but there are several other types of layouts. Each layout has a different purpose and order in which it displays its child View controls on the screen. Layouts are derived from android.view.ViewGroup.

The types of layouts built–in to the Android SDK framework include

- FrameLayout
- LinearLayout
- RelativeLayout
- TableLayout

All layouts, regardless of their type, have basic layout attributes. Layout attributes apply to any child View within that layout.You can set layout attributes at runtime programmatically, but ideally you set them in the XML layout files using the following syntax :

android:layout_attribute_name="value"

There are several layout attributes that all ViewGroup objects share. These include size attributes and margin attributes. You can find basic layout attributes in the ViewGroup.LayoutParams class. The margin attributes enable each child View within a layout to have padding on each side. Find these attributes in the ViewGroup.MarginLayoutParams classes. There are also a number of ViewGroup attributes for handling child View drawing bounds and animation settings.

Some of the important attributes shared by all ViewGroup subtypes are shown in Table 9.1.

**Table 9.1 : Important ViewGroup Attributes**

| Attribute Name | Applies To | Description | Value |
|---|---|---|---|
| `android: layout_height` | Parent view<br>Child view | Height of the view. Required attribute for child view controls in layouts. | Specific dimension value, `fill_parent`, or `wrap_content`.<br>The `match_parent` option is available in API Level 8+. |
| `android: layout_width` | Parent view<br>Child view | Width of the view. Required attribute for child view controls in layouts. | Specific dimension value, `fill_parent`, or `wrap_content`.<br>The `match_parent` option is available in API Level 8+. |
| `android: layout_margin` | Child view | Extra space on all sides of the view. | Specific dimension value. |

Here's an XML layout resource example of a LinearLayout set to the size of the screen, containing one TextView that is set to its full height and the width of the LinearLayout (and therefore the screen) :

```
<LinearLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<TextView
android:id="@+id/TextView01"
android:layout_height="fill_parent"
android:layout_width="fill_parent" />
</LinearLayout>
```

Here is an example of a Button object with some margins set via XML used in a layout resource file :

```
<Button
android:id="@+id/Button01"
android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
android:text="Press Me"
android:layout_marginRight="20px"
android:layout_marginTop="60px" />
```

Remember that layout elements can cover any rectangular space on the screen; it doesn't need to be the entire screen. Layouts can be nested within one another. This provides great flexibility when developers need to organize screen elements. It is common to start with a FrameLayout or LinearLayout as the parent layout for the entire screen and then organize individual screen elements inside the parent layout using whichever layout type is most appropriate.

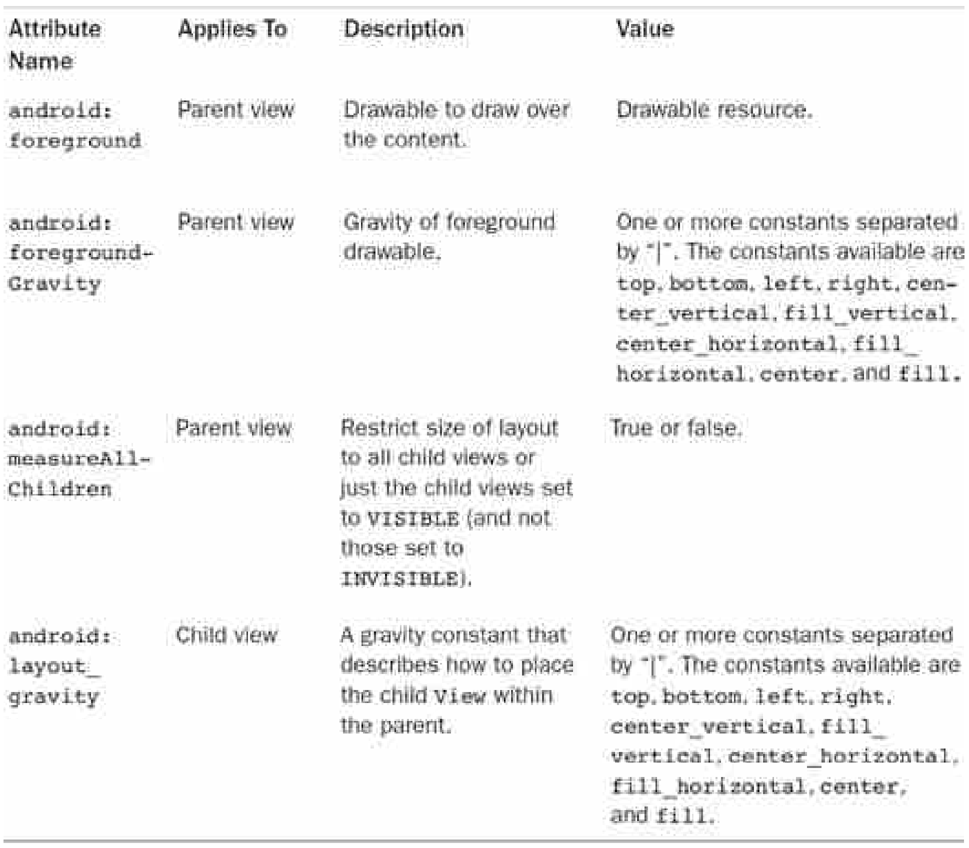
Now let's talk about each of the common layout types individually and how they differ from one another.

### 9.4.1 Using FrameLayout :

A FrameLayout view is designed to display a stack of child View items.You can add multiple views to this layout, but each View is drawn from the top–left corner of the layout. You can use this to show multiple images within the same region, as shown in Figure 9.2, and the layout is sized to the largest child View in the stack.

You can find the layout attributes available for FrameLayout child View objects in android.control.FrameLayout.LayoutParams. Table 9.2 describes some of the mportant attributes specific to FrameLayout views.

**Table 9.2 : Important FrameLayout View Attributes**



| Attribute Name | Applies To | Description | Value |
|---|---|---|---|
| `android:foreground` | Parent view | Drawable to draw over the content. | Drawable resource. |
| `android:foreground-Gravity` | Parent view | Gravity of foreground drawable. | One or more constants separated by "\|". The constants available are `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, and `fill`. |
| `android:measureAll-Children` | Parent view | Restrict size of layout to all child views or just the child views set to `VISIBLE` (and not those set to `INVISIBLE`). | True or false. |
| `android:layout_gravity` | Child view | A gravity constant that describes how to place the child `View` within the parent. | One or more constants separated by "\|". The constants available are `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, and `fill`. |

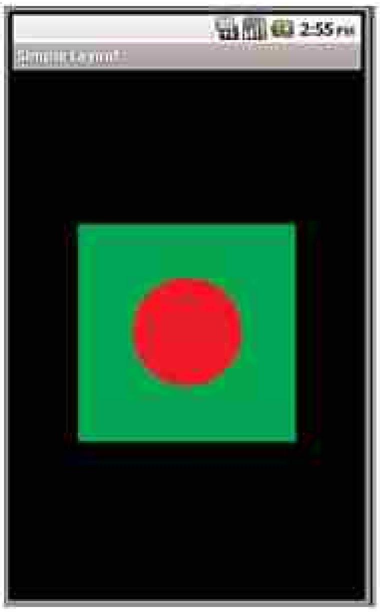


Figure 9.2 : An example of FrameLayout usage.

Here's an example of an XML layout resource with a FrameLayout and two child View objects, both ImageView objects.The green rectangle is drawn first and the red oval is drawn on top of it.The green rectangle is larger, so it defines the bounds of the

FrameLayout :

```
<FrameLayout xmlns:android=
http://schemas.android.com/apk/res/android
android:id="@+id/FrameLayout01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center">
<ImageView
android:id="@+id/ImageView01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/green_rect"
android:minHeight="200px"
android:minWidth="200px" />
<ImageView
android:id="@+id/ImageView02"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/red_oval"
android:minHeight="100px"
android:minWidth="100px"
android:layout_gravity="center" />
</FrameLayout>
```

### 9.4.2 Using LinearLayout :

A LinearLayout view organizes its child View objects in a single row, shown in Figure 9.3, or column, depending on whether its orientation attribute is set to horizontal or vertical.

This is a very handy layout method for creating forms.

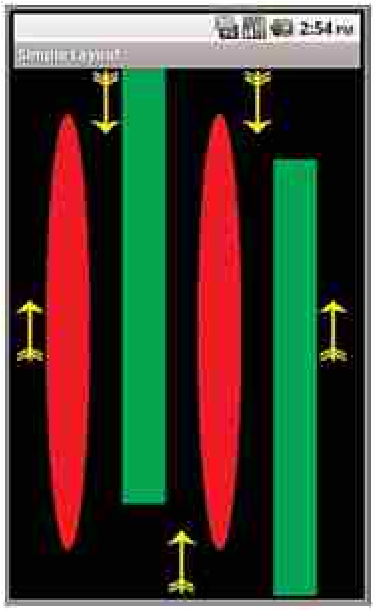


**Figure 9.3 : An example of LinearLayout (horizontal orientation).**

You can find the layout attributes available for LinearLayout child View objects in android.control.LinearLayout.LayoutParams. Table 9.3 describes some of the important attributes specific to LinearLayout views.

**Table 9.3 : Important LinearLayout View Attributes**

| Attribute Name | Applies To | Description | Value |
|---|---|---|---|
| `android:orientation` | Parent view | Layout is a single row (horizontal) or single column (vertical). | Horizontal or vertical. |
| `android:gravity` | Parent view | Gravity of child views within layout. | One or more constants separated by "\|". The constants available are `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, and `fill`. |
| `android:layout_gravity` | Child view | The gravity for a specific child view. Used for positioning of views. | One or more constants separated by "\|". The constants available are `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, and `fill`. |
| `android:layout_weight` | Child view | The weight for a specific child view. Used to provide ratio of screen space used within the parent control. | The sum of values across all child views in a parent view must equal 1. For example, one child control might have a value of `.3` and another have a value of `.7`. |

### 9.4.3 Using RelativeLayout :

The RelativeLayout view enables you to specify where the child view controls are in relation to each other. For instance, you can set a child View to be positioned "above" or "below" or "to the left of" or "to the right of" another View, referred to by its unique identifier. You can also align child View objects relative to one another or the parent layout edges. Combining RelativeLayout attributes can simplify creating interesting user interfaces without resorting to multiple layout groups to achieve a desired effect. Figure 9.4 shows how each of the button controls is relative to each other.

You can find the layout attributes available for RelativeLayout child View objects in android.control.RelativeLayout.LayoutParams. Table 9.4 describes some of the important attributes specific to RelativeLayout views.

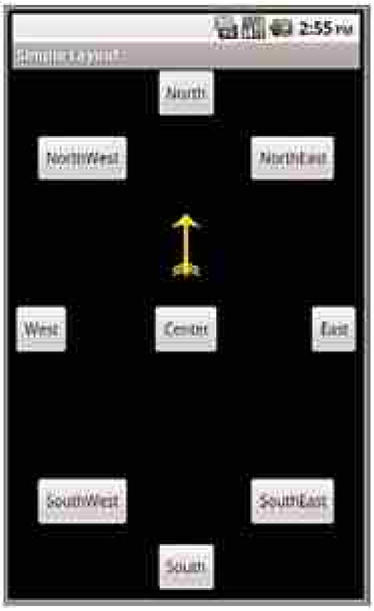


**Figure 9.4 : An example of RelativeLayout usage.**

**Table 9.4 : Important RelativeLayout View Attributes**

| Attribute Name | Applies To | Description | Value |
|---|---|---|---|
| `android:gravity` | Parent view | Gravity of child views within layout. | One or more constants separated by "\|". The constants available are `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, and `fill`. |
| `android:layout_centerInParent` | Child view | Centers child view horizontally and vertically within parent view. | True or false. |
| `android:layout_centerHorizontal` | Child view | Centers child view horizontally within parent view. | True or false. |
| `android:layout_centerVertical` | Child view | Centers child view vertically within parent view. | True or false. |
| `android:layout_alignParentTop` | Child view | Aligns child view with top edge of parent view. | True or false. |
| `android:layout_alignParentBottom` | Child view | Aligns child view with bottom edge of parent view. | True or false. |

| | | | |
|---|---|---|---|
| android: layout_ alignParentLeft | Child view | Aligns child view with left edge of parent view. | True or false. |
| android: layout_ alignParentRight | Child view | Aligns child view with right edge of parent view. | True or false. |
| android: layout_ alignRight | Child view | Aligns child view with right edge of another child view, specified by ID. | A view ID; for example, @id/Button1 |
| android: layout_ alignLeft | Child view | Aligns child view with left edge of another child view, specified by ID. | A view ID; for example, @id/Button1 |
| android: layout_ alignTop | Child view | Aligns child view with top edge of another child view, specified by ID. | A view ID; for example, @id/Button1 |
| android: layout_ alignBottom | Child view | Aligns child view with bottom edge of another child view, specified by ID. | A view ID; for example, @id/Button1 |
| android: layout_ above | Child view | Positions bottom edge of child view above another child view, specified by ID. | A view ID; for example, @id/Button1 |
| android: layout_ below | Child view | Positions top edge of child view below another child view, specified by ID. | A view ID; for example, @id/Button1 |
| android: layout_ toLeftOf | Child view | Positions right edge of child view to the left of another child view, specified by ID. | A view ID; for example, @id/Button1 |
| android: layout_ toRightOf | Child view | Positions left edge of child view to the right of another child view, specified by ID. | A view ID; for example, @id/Button1 |

Here's an example of an XML layout resource with a RelativeLayout and two child View objects, a Button object aligned relative to its parent, and an ImageView aligned and positioned relative to the Button (and the parent) :

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
android:id="@+id/RelativeLayout01"
android:layout_height="fill_parent"
android:layout_width="fill_parent">
<Button
android:id="@+id/ButtonCenter"
android:text="Center"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerInParent="true" />
<ImageView
android:id="@+id/ImageView01"
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_above="@id/ButtonCenter"
android:layout_centerHorizontal="true"
android:src="@drawable/arrow" />
</RelativeLayout>
```

### 9.4.4 Using TableLayout :

A TableLayout view organizes children into rows, as shown in Figure 9.5. You add individual View objects within each row of the table using a TableRow layout View (which is basically a horizontally oriented LinearLayout) for each row of the table. Each column of the TableRow can contain one View (or layout with child View objects). You place View items added to a TableRow in columns in the order they are added.You can specify the column number (zero–based) to skip columns as necessary (the bottom row shown in Figure 9.5 demonstrates this); otherwise, the View object is put in the next column to the right. Columns scale to the size of the largest View of that column.You can also include normal View objects instead of TableRow elements, if you want the View to take up an entire row.

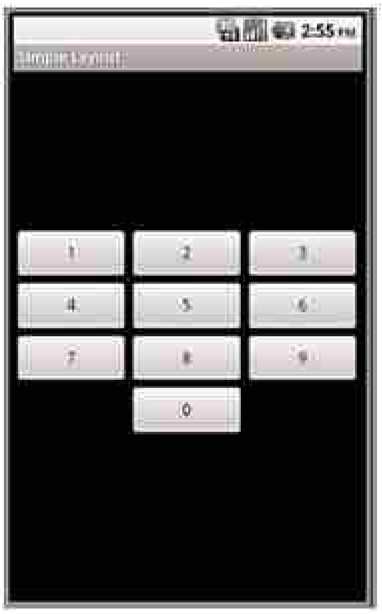


**Figure 9.5 : An example of TableLayout usage.**

You can find the layout attributes available for TableLayout child View objects in android.control.TableLayout.LayoutParams. You can find the layout attributes available for TableRow child View objects in android.control.TableRow. LayoutParams.

Table 9.5 describes some of the important attributes specific to TableLayout View objects.

**Table 9.5 : Important TableLayout and TableRow View Attributes**

| Attribute Name | Applies To | Description | Value |
|---|---|---|---|
| `android: collapseColumns` | `TableLayout` | A comma-delimited list of column indices to collapse (0-based) | String or string resource. For example, `"0,1,3,5"` |
| `android: shrinkColumns` | `TableLayout` | A comma-delimited list of column indices to shrink (0-based) | String or string resource. Use `"*"` for all columns. For example, `"0,1,3,5"` |
| `andriod: stretchColumns` | `TableLayout` | A comma-delimited list of column indices to stretch (0-based) | String or string resource. Use `"*"` for all columns. For example, `"0,1,3,5"` |
| `android: layout_column` | `TableRow` child view | Index of column this child view should be displayed in (0-based) | Integer or integer re-source. For example, 1 |
| `android: layout_span` | `TableRow` child view | Number of columns this child view should span across | Integer or integer re-source greater than or equal to 1. For example, 3 |

Here's an example of an XML layout resource with a TableLayout with two rows (two TableRow child objects). The TableLayout is set to stretch the columns to the size of the screen width. The first TableRow has three columns; each cell has a Button object. The second TableRow puts only one Button view into the second column explicitly :

```
<TableLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
android:id="@+id/TableLayout01"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:stretchColumns="*">
<TableRow
android:id="@+id/TableRow01">
<Button
android:id=
"@+id/ButtonLeft" android:text="Left Door" />
<Button
android:id="@+id/ButtonMiddle"
android:text="Middle Door" />
<Button
android:id="@+id/ButtonRight"
android:text="Right Door" />
</TableRow>
<TableRow
android:id="@+id/TableRow02">
```

```
<Button
android:id="@+id/ButtonBack"
android:text="Go Back"
android:layout_column="1" />
</TableRow>
</TableLayout>
```

❑ **Check Your Progress :**

1. What is the default value of the orientation attribute in LinearLayout ?

   (A) Vertical (B) Relative (C) Absolute (D) Horizontal

2. What is the nine–patch images tool in android?

   (A) It is used to change the bitmap images into nine sections

   (B) It is used to change the image into various parts

   (C) It is used to decorate the image

   (D) None of the Above

3. The ________ view enables you to specify where the child view controls are in relation to each other.

   (A) LinearLayout (B) TableRow

   (C) RelativeLayout (D) CardLayout

4. A ________ view is designed to display a stack of child View items.

   (A) RelativeLayout (B) TableRow

   (C) LinearLayout (D) FrameLayout

5. ________ is not in a layout available in android.

   (A) Relative (B) Absolute (C) Linear (D) Table

## 9.5 Let Us Sum Up :

In this unit we learnt about various android screen elements and various layout available for the proper design and look out of the application. We discussed in depth regarding the implementation of all kind of layout and screen elements.

## 9.6 Answers for Check Your Progress :

**1.** (D) **2.** (A) **3.** (C) **4.** (D) **5.** (B)

## 9.7 Glossary :

**1.** **SDK :** Software Development Kit

**2.** **AVD :** Android Virtual Device

**3.** **DDMS :** Dalvik Debug Monitor Server

**4.** **ADB :** Android Debug Bridge

## 9.8 Assignment :

1. Which method and attribute is used to set image of ImageView ?
2. For what purpose Chronometer is used? Explain the Chronometer object's format attribute and explain the different methods associated with Chronometer

3. Which method and attribute are used to set image of ImageView ?
4. Explain various Layout available in Android.

## 9.9 Activities :

1. Identify the various android controls and arrange them with proper layouts available in the android.

## 9.10 Case Study :

Analysis of the system and design the user interface with proper layout and look and feel.

## 9.11 Further Reading :

Lauren Darcey and Shane Conder, "Android Wireless Application Development", Pearson Education, 2nd ed. (2011).

# Unit 10 DRAWING AND WORKING WITH ANIMATION

## UNIT STRUCTURE

## 10.0 Learning Objectives :

- To learn working with drawing & Animation
- To learn & understand the use of Canvas and Paint
- To understand the work with bitmap and shapes
- To understand how to set and apply animation in Android Application

## 10.1 Introduction :

This unit talks about the drawing and animation features built into Android, including creating custom View classes and working with Canvas and Paint to draw shapes and text. We also talk about animating objects on the screen in a variety of ways. In previous unit, we talk about layouts and the various View classes available in Android to make screen design simple and efficient. Now we must think at a slightly lower level and talk about drawing objects on the screen. With Android, we can display images such as PNG and JPG graphics, as well as text and primitive shapes to the screen. We can paint these items with various colors, styles, or gradients and modify them using standard image transforms. We can even animate objects to give the illusion of motion.

## 10.2 Drawing on the Screen :

### 10.2.1 Working with Canvases and Paints :

To draw to the screen, you need a valid Canvas object.Typically we get a valid Canvas object by extending the View class for our own purposes and implementing the onDraw() method.

For example, here's a simple View subclass called ViewWithRedDot. We override the onDraw() method to dictate what the View looks like; in this case, it draws a red circle on a black background.

```
private static class ViewWithRedDot extends View {
public ViewWithRedDot(Context context) {
super(context);
}
@Override
protected void onDraw(Canvas canvas) {
canvas.drawColor(Color.BLACK);
Paint circlePaint = new Paint();
circlePaint.setColor(Color.RED);
canvas.drawCircle(canvas.getWidth()/2,
canvas.getHeight()/2,
canvas.getWidth()/3, circlePaint);
}
}
```

We can then use this View like any other layout. For example,we might override the onCreate() method in our Activity with the following :

```
setContentView(new ViewWithRedDot(this));
```

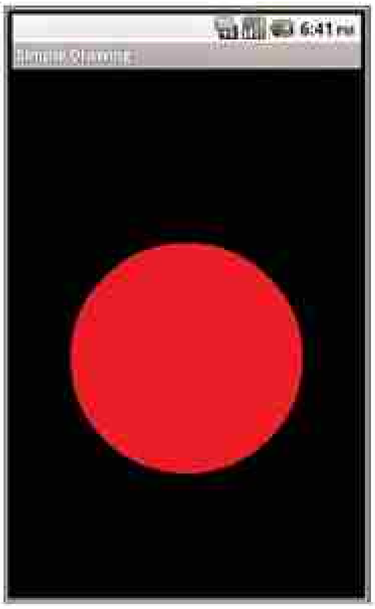


**Figure 10.1 : The ViewWithRedDot view draws a red circle on a black canvas background.**

### Understanding the Canvas :

The Canvas (android.graphics.Canvas) object holds the draw calls, in order, for a rectangle of space. There are methods available for drawing images, text, shapes, and support for clipping regions. The dimensions of the Canvas are bound by the container view.You can retrieve the size of the Canvas using the getHeight() and getWidth() methods.

### Understanding the Paint :

In Android, the Paint (android.graphics.Paint) object stores far more than a color. The Paint class encapsulates the style and complex color and rendering information, which can be applied to a drawable like a graphic, shape, or piece of text in a given Typeface.

### Working with Paint Color :

You can set the color of the Paint using the setColor() method. Standard colors are predefined within the android.graphics. Color class. For example, the following code sets the paint color to red :

```
Paint redPaint = new Paint();
redPaint.setColor(Color.RED);
```

### Working with Paint Antialiasing :

Antialiasing makes many graphics–whether they are shapes or typefaces–look smoother on the screen. This property is set within the Paint of an object. For example, the following code instantiates a Paint object with antialiasing enabled :

```
Paint aliasedPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
```

### Working with Paint Styles :

Paint style controls how an object is filled with color. For example, the following code instantiates a Paint object and sets the Style to STROKE, which signifies that the object should be painted as a line drawing and not filled (the default) :

```
Paint linePaint = new Paint();
linePaint.setStyle(Paint.Style.STROKE);
```

### Working with Paint Gradients :

You can create a gradient of colors using one of the gradient subclasses.The different gradient classes (see Figure 10.2), including LinearGradient, RadialGradient, and SweepGradient, are available under the superclass android.graphics.Shader.

All gradients need at least two colors–a start color and an end color–but might contain any number of colors in an array. The different types of gradients are differentiated by the direction in which the gradient "flows." Gradients can be set to mirror and repeat as necessary.

You can set the Paint gradient using the setShader() method.

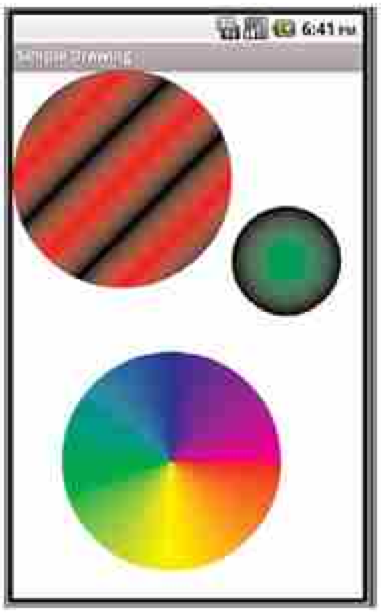


**Figure 10.2 : An example of a LinearGradient (top), a RadialGradient (right), and a SweepGradient (bottom).**

**Working with Linear Gradients :**

A linear gradient is one that changes colors along a single straight line. The top–left circle in Figure 10.2 is a linear gradient between black and red, which is mirrored.

You can achieve this by creating a LinearGradient and setting the Paint method setShader() before drawing on a Canvas, as follows :

```
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.LinearGradient;
import android.graphics.Paint;
import android.graphics.Shader;
...
Paint circlePaint =
new Paint(Paint.ANTI_ALIAS_FLAG);
LinearGradient linGrad =
new LinearGradient(0, 0, 25, 25,
Color.RED, Color.BLACK,
Shader.TileMode.MIRROR);
circlePaint.setShader(linGrad);
canvas.drawCircle(100, 100, 100, circlePaint);
```

**Working with Radial Gradients :**

A radial gradient is one that changes colors starting at a single point and radiating outward in a circle.The smaller circle on the right in Figure 10.2 is a radial gradient between green and black.

You can achieve this by creating a RadialGradient and setting the Paint method setShader() before drawing on a Canvas, as follows :

```
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.RadialGradient;
import android.graphics.Paint;
import android.graphics.Shader;
...
Paint circlePaint =
new Paint(Paint.ANTI_ALIAS_FLAG);
RadialGradient radGrad = new RadialGradient(250,
175, 50, Color.GREEN, Color.BLACK,
Shader.TileMode.MIRROR);
circlePaint.setShader(radGrad);
canvas.drawCircle(250, 175, 50, circlePaint);
```

**Working with Sweep Gradients :**

A sweep gradient is one that changes colors using slices of a pie. This type of gradient is often used for a color chooser. The large circle at the bottom of Figure 10.2 is a sweep gradient between red, yellow, green, blue, and magenta.

You can achieve this by creating a SweepGradient and setting the Paint method setShader() before drawing on a Canvas, as follows :

```
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.SweepGradient;
import android.graphics.Paint;
import android.graphics.Shader;
...
Paint circlePaint =
new Paint(Paint.ANTI_ALIAS_FLAG);
SweepGradient sweepGrad = new
SweepGradient(canvas.getWidth()-175,
canvas.getHeight()-175,
new int[] { Color.RED, Color.YELLOW, Color.GREEN,
Color.BLUE, Color.MAGENTA }, null);
circlePaint.setShader(sweepGrad);
canvas.drawCircle(canvas.getWidth()-175,
canvas.getHeight()-175, 100,
circlePaint);
```

**Working with Paint Utilities for Drawing Text :**

The Paint class includes a number of utilities and features for rendering text to the screen in different typefaces and styles. Now is a great time to start drawing some text to the screen.

### 10.2.2 Working with Text :

Android provides several default font typefaces and styles. Applications can also use custom fonts by including font files as application assets and loading them using the Asset Manager, much as one would use resources

**Using Default Fonts and Typefaces :**

By default,Android uses the Sans Serif typeface, but Monospace and Serif typefaces are also available. The following code excerpt draws some antialiased text in the default typeface (Sans Serif) to a Canvas :

```
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Typeface;
...
Paint mPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
Typeface mType;
mPaint.setTextSize(16);
mPaint.setTypeface(null);
canvas.drawText("Default Typeface", 20, 20, mPaint);
```

You can instead load a different typeface, such as Monotype:

```
Typeface mType = Typeface.create(Typeface.MONOSPACE,
Typeface.NORMAL);
```

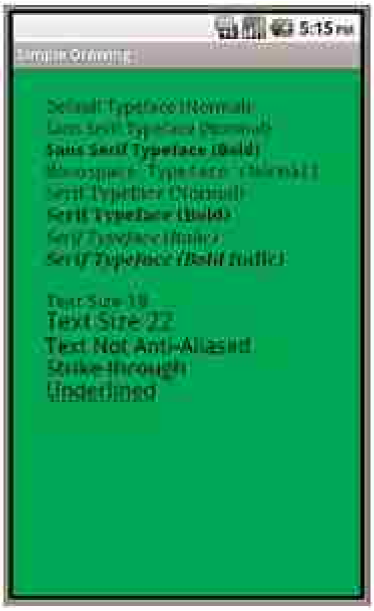
Perhaps you would prefer italic text, in which case you can simply set the style of the typeface and the font family :

```
Typeface mType = Typeface.create(Typeface.SERIF,
Typeface.ITALIC);
```

You can set certain properties of a typeface such as antialiasing, underlining, and strikethrough using the setFlags() method of the Paint object :

mPaint.setFlags(Paint.UNDERLINE_TEXT_FLAG);

Figure 10.3 shows some of the Typeface families and styles available by default on Android.


5:15 PM

Simple Drawing

Default Typeface (Normal)
Sans Serif Typeface (Normal)
**Sans Serif Typeface (Bold)**
Monospace Typeface (Normal)
Serif Typeface (Normal)
**Serif Typeface (Bold)**
*Serif Typeface (Italic)*
***Serif Typeface (Bold Italic)***

Text Size 18
Text Size 22
Text Not Anti-Aliased
~~Strike-through~~
Underlined

**Figure 10.3 : Some typefaces and typeface styles available on Android.**

**Using Custom Typefaces :**

You can easily use custom typefaces with your application by including the font file as an application asset and loading it on demand. Fonts might be used for a custom look– and feel, for implementing language symbols that are not supported natively, or for custom symbols.

For example, you might want to use a handy chess font to implement a simple, scalable chess game. A chess font includes every symbol needed to implement a chessboard, including the board and the pieces. Hans Bodlaender has kindly provided a free chess font called Chess Utrecht. Using the Chess Utrecht font, the letter Q draws a black queen on a white square, whereas a q draws a white queen on a white square, and so on. This nifty font is available at www.chessvariants.com/d.font/utrecht.html as chess1.ttf.

To use a custom font, such as Chess Utrecht, simply download the font from the website and copy the chess1.ttf file from your hard drive to the project directory /assets/fonts/chess1.ttf.

Now you can load the Typeface object programmatically much as you would any resource :

```
import android.graphics.Typeface;
import android.graphics.Color;
import android.graphics.Paint;
...
Paint mPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
Typeface mType =
Typeface.createFromAsset(getContext().getAssets(),
"fonts/chess1.ttf");
```

You can then use the Chess Utrecht typeface to "draw" a chessboard (see Figure 10.4) using the appropriate character sequences.

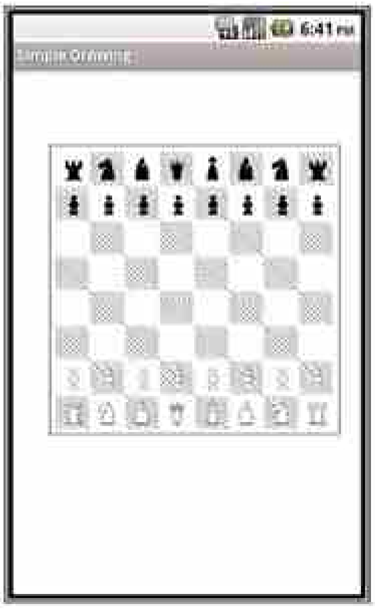


**Figure 10.4 : Using the Chess Utrecht font to draw a chessboard.**

**Measuring Text Screen Requirements :**

You can measure how large text with a given Paint is and how big of a rectangle you need to encompass it using the measureText() and getTextBounds() methods.

**10.2.3 Working with Bitmaps :**

You can find lots of goodies for working with graphics such as bitmaps (including NinePatch) in the android.graphics package. The core class for bitmaps is android.graphics.Bitmap.

**Drawing Bitmap Graphics on a Canvas :**

You can draw bitmaps onto a valid Canvas, such as within the onDraw() method of a View, using one of the drawBitmap() methods. For example, the following code loads a Bitmap resource and draws it on a canvas :

```
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
...
Bitmap pic =
BitmapFactory.decodeResource(getResources(),
R.drawable.bluejay);
canvas.drawBitmap(pic, 0, 0, null);
```

**Scaling Bitmap Graphics :**

Perhaps you want to scale your graphic to a smaller size. In this case, you can use the createScaledBitmap() method, like this :

Bitmap sm = Bitmap.createScaledBitmap(pic, 50, 75, false);

You can preserve the aspect ratio of the Bitmap by checking the getWidth() and getHeight() methods and scaling appropriately.

**Transforming Bitmaps Using Matrixes :**

You can use the helpful Matrix class to perform transformations on a Bitmap graphic (see Figure 10.5). Use the Matrix class to perform tasks such as mirroring and rotating graphics, among other actions.

The following code uses the createBitmap() method to generate a new Bitmap that is a mirror of an existing Bitmap called pic:

```
import android.graphics.Bitmap;
import android.graphics.Matrix;
...
Matrix mirrorMatrix = new Matrix();
mirrorMatrix.preScale(-1, 1);
Bitmap mirrorPic = Bitmap.createBitmap(pic, 0, 0,
pic.getWidth(), pic.getHeight(), mirrorMatrix, false);
```

You can perform a 30–degree rotation in addition to mirroring by using this Matrix instead :

```
Matrix mirrorAndTilt30 = new Matrix();
mirrorAndTilt30.preRotate(30);
mirrorAndTilt30.preScale(-1, 1);
```

You can see the results of different combinations of tilt and mirror Matrix transforms in Figure 10.5. When you're no longer using a Bitmap, you can free its memory using the recycle() method :

pic.recycle();

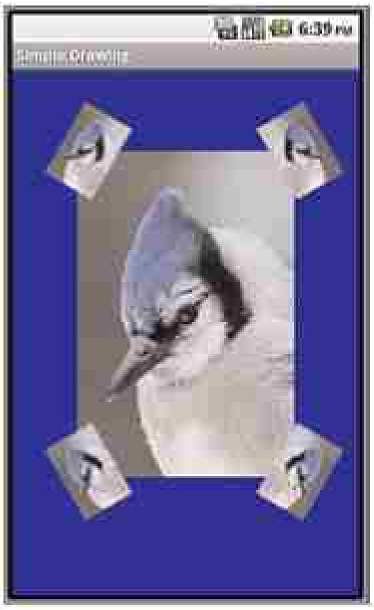


**Figure 10.5 : A single–source bitmap : scaled, tilted, and mirrored using Android Bitmap classes.**

There are a variety of other Bitmap effects and utilities available as part of the Android SDK, but they are numerous and beyond the scope of this book. See the android.graphics package for more details.

### 10.2.4 Working with Shapes :

You can define and draw primitive shapes such as rectangles and ovals using the ShapeDrawable class in conjunction with a variety of specialized Shape classes. You can define Paintable drawables as XML resource files, but more often, especially with more complex shapes, this is done programmatically.

**Defining Shape Drawables as XML Resources :**

We show you how to define primitive shapes such as rectangles using specially formatted XML files within the /res/drawable/resource directory.

The following resource file called /res/drawable/green_rect.xml describes a simple, green rectangle shape drawable :

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android=
"http://schemas.android.com/apk/res/android"
android:shape="rectangle">
<solid android:color="#0f0"/>
</shape>
```

You can then load the shape resource and set it as the Drawable as follows :

```
ImageView iView =
(ImageView)findViewById(R.id.ImageView1);
iView.setImageResource(R.drawable.green_rect);
```

You should note that many Paint properties can be set via XML as part of the Shape definition. For example, the following Oval shape is defined with a linear gradient (red to white) and stroke style information :

```
<?xml version="1.0" encoding="utf-8"?>
<shapexmlns:android=
"http://schemas.android.com/apk/res/android"
android:shape="oval">
<solid android:color="#f00"/>
<gradient android:startColor="#f00"
android:endColor="#fff"
android:angle="180"/>
<stroke android:width="3dp" android:color="#00f"
android:dashWidth="5dp" android:dashGap="3dp"/>
</shape>
```

**Defining Shape Drawables Programmatically :**

You can also define these ShapeDrawable instances programmatically. The different shapes are available as classes within the android.graphics.drawable. shapes package. For example, you can programmatically define the aforementioned green rectangle as follows :

```
import android.graphics.drawable.ShapeDrawable;
import android.graphics.drawable.shapes.RectShape;
...
ShapeDrawable rect =
new ShapeDrawable(new RectShape());
rect.getPaint().setColor(Color.GREEN);
```

You can then set the Drawable for the ImageView directly :

```
ImageView iView =
(ImageView)findViewById(R.id.ImageView1);
iView.setImageDrawable(rect);
```

The resulting green rectangle is shown in Figure 10.6.

**Drawing Different Shapes :**

Some of the different shapes available within the android.graphics.drawable.shapes package include Rectangles (and squares)

- Rectangles with rounded corners
- Ovals (and circles)
- Arcs and lines
- Other shapes defined as paths

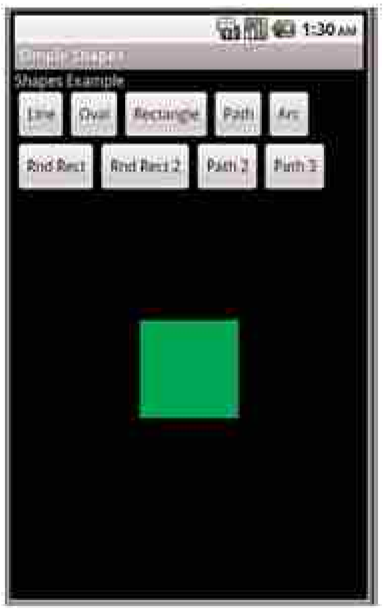


**Figure 10.6 : A green rectangle.**

You can create and use these shapes as Drawable resources directly within ImageView views, or you can find corresponding methods for creating these primitive shapes within a Canvas.

**Drawing Rectangles and Squares :**

Drawing rectangles and squares (rectangles with equal height/width values) is simply a matter of creating a ShapeDrawable from a RectShape object. The RectShape object has no dimensions but is bound by the container object– in this case, the ShapeDrawable.

You can set some basic properties of the ShapeDrawable, such as the Paint color and the default size.

For example, here we create a magenta–colored rectangle that is 100–pixels long and 2–pixels wide, which looks like a straight, horizontal line. We then set the shape as the drawable for an ImageView so the shape can be displayed :

```
import android.graphics.drawable.ShapeDrawable;
import android.graphics.drawable.shapes.RectShape;
...
ShapeDrawable rect =
new ShapeDrawable(new RectShape());
rect.setIntrinsicHeight(2);
rect.setIntrinsicWidth(100);
rect.getPaint().setColor(Color.MAGENTA);
ImageView iView =
(ImageView)findViewById(R.id.ImageView1);
iView.setImageDrawable(rect);
```

**Drawing Rectangles with Rounded Corners :**

You can create rectangles with rounded corners, which can be nice for making custom buttons. Simply create a ShapeDrawable from a RoundRectShape object. The RoundRectShape requires an array of eight float values, which signify the radii of the rounded corners. For example, the following creates a simple cyan–colored, rounded–corner rectangle :

```
import android.graphics.drawable.ShapeDrawable;
import android.graphics.drawable.shapes.
RoundRectShape;
...
ShapeDrawable rndrect = new ShapeDrawable(
new RoundRectShape( new float[]
{ 5, 5, 5, 5, 5, 5, 5, 5 }, null, null));
rndrect.setIntrinsicHeight(50);
rndrect.setIntrinsicWidth(100);
rndrect.getPaint().setColor(Color.CYAN);
ImageView iView =
(ImageView)findViewById(R.id.ImageView1);
iView.setImageDrawable(rndrect);
```

The resulting round–corner rectangle is shown in Figure 10.7.

You can also specify an inner–rounded rectangle within the outer rectangle, if you so choose.The following creates an inner rectangle with rounded edges within the outer white rectangle with rounded edges :

```
import android.graphics.drawable.ShapeDrawable;
import android.graphics.drawable.shapes.
RoundRectShape;
...
```

```
float[] outerRadii =
new float[]{ 6, 6, 6, 6, 6, 6, 6, 6 };
RectF insetRectangle =
new RectF(8, 8, 8, 8);
float[] innerRadii =
new float[]{ 6, 6, 6, 6, 6, 6, 6, 6 };
ShapeDrawable rndrect = new ShapeDrawable(
new RoundRectShape(
outerRadii, insetRectangle, innerRadii));
rndrect.setIntrinsicHeight(50);
rndrect.setIntrinsicWidth(100);
rndrect.getPaint().setColor(Color.WHITE);
ImageView iView =
(ImageView)findViewById(R.id.ImageView1);
iView.setImageDrawable(rndrect);
```

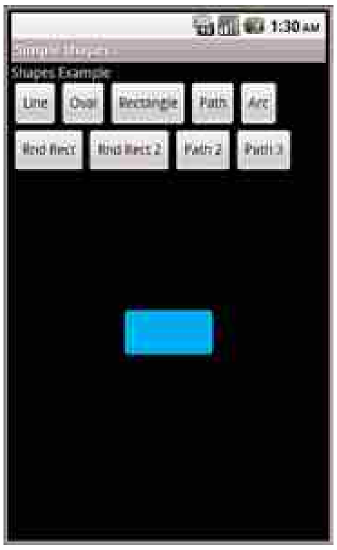


Figure 10.7 : A cyan rectangle with rounded corners.

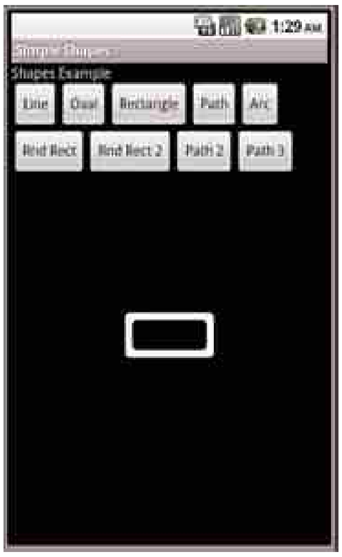


Figure 10.8 : A white rectangle with rounded corners, with an inset rounded rectangle.

**Drawing Ovals and Circles :**

You can create ovals and circles (which are ovals with equal height/width values) by creating a ShapeDrawable using an OvalShape object. The OvalShape object has no dimensions but is bound by the container object–in this case, the ShapeDrawable. You can set some basic properties of the ShapeDrawable, such as the Paint color and the default size.

For example, here we create a red oval that is 40–pixels high and 100–pixels wide, which looks like a Frisbee :

```
import android.graphics.drawable.ShapeDrawable;
import android.graphics.drawable.shapes.OvalShape;
...
ShapeDrawable oval =
new ShapeDrawable(new OvalShape());
oval.setIntrinsicHeight(40);
oval.setIntrinsicWidth(100);
oval.getPaint().setColor(Color.RED);
ImageView iView =
(ImageView)findViewById(R.id.ImageView1);
iView.setImageDrawable(oval);
```

The resulting red oval is shown in Figure 10.9.

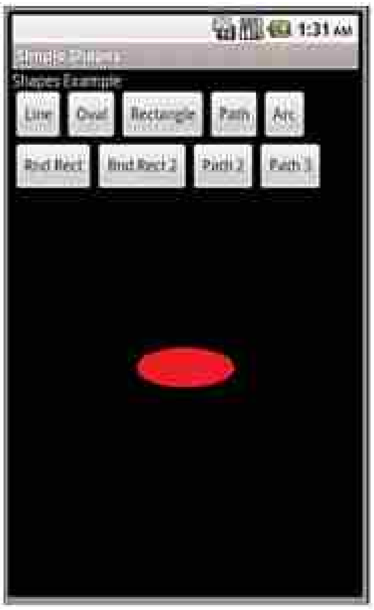


**Figure 10.9 : A red oval.**

**Drawing Arcs :**

You can draw arcs, which look like pie charts or Pac–Man, depending on the sweep angle you specify. You can create arcs by creating a ShapeDrawable by using an ArcShape object.

The ArcShape object requires two parameters: a startAngle and a sweepAngle. The startAngle begins at 3 o'clock. Positive sweepAngle values sweep clockwise; negative values sweep counterclockwise. You can create a circle by using the values 0 and 360.

The following code creates an arc that looks like a magenta Pac–Man :

```
import android.graphics.drawable.ShapeDrawable;
import android.graphics.drawable.shapes.ArcShape;
...
ShapeDrawable pacMan =
new ShapeDrawable(new ArcShape(0, 345));
pacMan.setIntrinsicHeight(100);
pacMan.setIntrinsicWidth(100);
pacMan.getPaint().setColor(Color.MAGENTA);
ImageView iView =
(ImageView)findViewById(R.id.ImageView1);
iView.setImageDrawable(pacMan);
```

The resulting arc is shown in Figure 10.10.

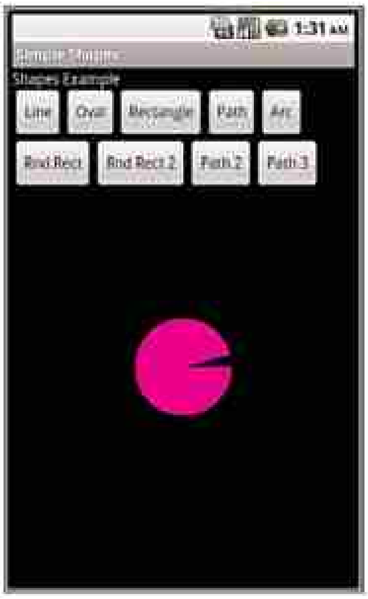


**Figure 10.10 : A magenta arc of 345 degrees (resembling Pac–Man).**

**Drawing Paths :**

You can specify any shape you want by breaking it down into a series of points along a path. The android.graphics.Path class encapsulates a series of lines and curves that make up some larger shape.

For example, the following Path defines a rough five–point star shape :

```
import android.graphics.Path;
...
Path p = new Path();
p.moveTo(50, 0);
p.lineTo(25,100);
p.lineTo(100,50);
p.lineTo(0,50);
p.lineTo(75,100);
p.lineTo(50,0);
```

You can then encapsulate this star Path in a PathShape, create a ShapeDrawable, and paint it yellow.

```
import android.graphics.drawable.ShapeDrawable;
import android.graphics.drawable.shapes.PathShape;
...
ShapeDrawable star =
new ShapeDrawable(new PathShape(p, 100, 100));
star.setIntrinsicHeight(100);
star.setIntrinsicWidth(100);
star.getPaint().setColor(Color.YELLOW);
```

By default, this generates a star shape filled with the Paint color yellow (see Figure 10.11).

Or, you can set the Paint style to Stroke for a line drawing of a star.

star.getPaint().setStyle(Paint.Style.STROKE);

The resulting star would look something like Figure 10.12.

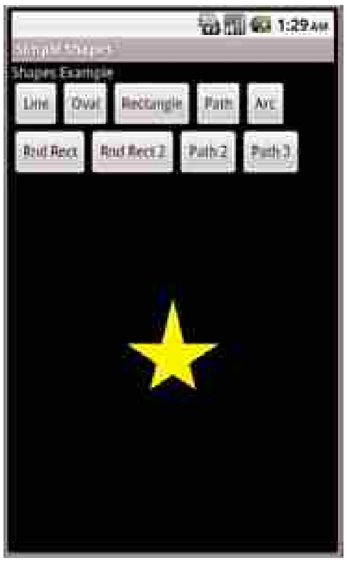


**Figure 10.11 : A yellow star.**

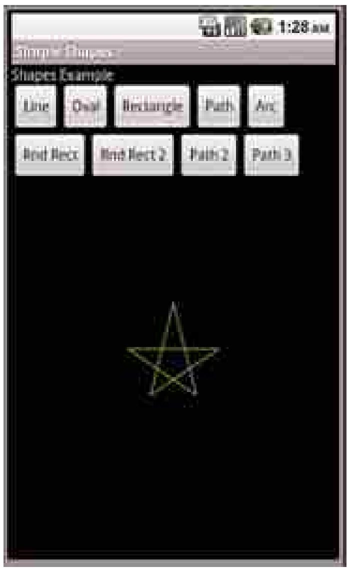


**Figure 10.12 : A yellow star using the stroke style of Paint.**

## 10.3 Working with Animation :

The Android platform supports three types of graphics animation :

- Animated GIF images
- Frame–by–frame animation
- Tweened animation

Animated GIFs store the animation frames within the image, and you simply include these GIFs like any other graphic drawable resource. For frame–by–frame animation, the developer must provide all graphics frames of the animation. However, with tweened animation, only a single graphic is needed, upon which transforms can be programmatically applied.

### 10.3.1 Working with Frame–by–Frame Animation :

You can think of frame–by–frame animation as a digital flipbook in which a series of similar images display on the screen in a sequence, each subtly different from the last. When you display these images quickly, they give the illusion of movement. This technique is called frame–by–frame animation and is often used on the Web in the form of animated GIF images.

Frame–by–frame animation is best used for complicated graphics transformations that are not easily implemented programmatically.

For example,we can create the illusion of a genie juggling gifts using a sequence of three images, as shown in Figure 10.13.

**Figure 10.13 : Three frames for an animation of a genie juggling.**

In each frame, the genie remains fixed, but the gifts are repositioned slightly. The smoothness of the animation is controlled by providing an adequate number of frames and choosing the appropriate speed on which to swap them.

The following code demonstrates how to load three Bitmap resources (our three genie frames) and create an AnimationDrawable. We then set the AnimationDrawable as the background resource of an ImageView and start the animation :

```
ImageView img =
(ImageView)findViewById(R.id.ImageView1);
BitmapDrawable frame1 =
(BitmapDrawable)getResources().
getDrawable(R.drawable.f1);
BitmapDrawable frame2 =
(BitmapDrawable)getResources().
```

```
getDrawable(R.drawable.f2);
BitmapDrawable frame3 =
(BitmapDrawable)getResources().
getDrawable(R.drawable.f3);
int reasonableDuration = 250;
AnimationDrawable mAnimation =
new AnimationDrawable();
mAnimation.addFrame(frame1, reasonableDuration);
mAnimation.addFrame(frame2, reasonableDuration);
mAnimation.addFrame(frame3, reasonableDuration);
img.setBackgroundDrawable(mAnimation);
```

To name the animation loop continuously, we can call the setOneShot() method :

```
mAnimation.setOneShot(false);
```

To begin the animation,we call the start() method:

```
mAnimation.start();
```

We can end our animation at any time using the stop() method:

```
mAnimation.stop();
```

Although we used an ImageView background in this example, you can use a variety of different View widgets for animations. For example, you can instead use the ImageSwitcher view and change the displayed Drawable resource using a timer. This sort of operation is best done on a separate thread.The resulting animation might look something like Figure 10.14 you just have to imagine it moving.

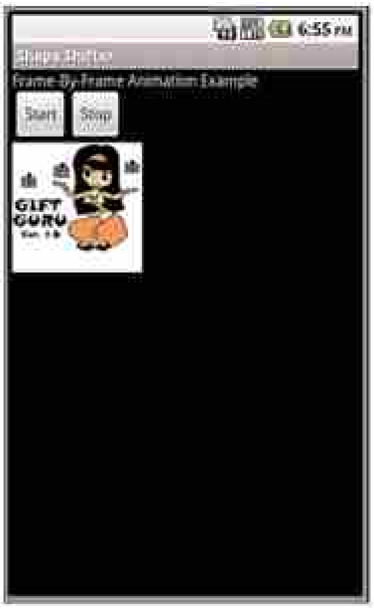


**Figure 10.14 : The genie animation in the Android emulator.**

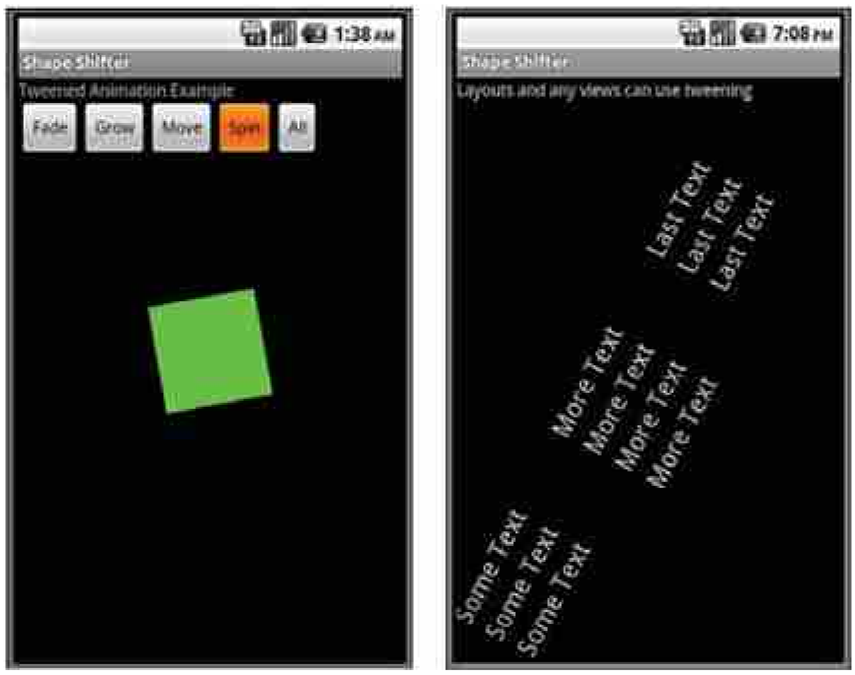


**Figure 10.15 : Rotating a green rectangle shape drawable (left) and a TableLayout (right).**

**10.3.2 Working with Tweened Animations :**

With tweened animation, you can provide a single Drawable resource– it is a Bitmap graphic (see Figure 10.15, left), a ShapeDrawable, a TextView (see Figure 10.15, right), or any other type of View object–and the intermediate frames of the animation are rendered by the system.Android provides tweening support for several common image transformations, including alpha, rotate, scale, and translate animations.You can apply tweened animation transformations to any View, whether it is an ImageView with a Bitmap or shape Drawable, or a layout such as a TableLayout.

**Defining Tweening Transformations :**

You can define tweening transformations as XML resource files or programmatically. All tweened animations share some common properties, including when to start, how long to animate, and whether to return to the starting state upon completion.

**Defining Tweened Animations as XML Resources :**

In Chapter 6, we showed you how to store animation sequences as specially formatted

XML files within the /res/anim/ resource directory. For example, the following resource file called /res/anim/spin.xml describes a simple five–second rotation :

```
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android
= "http://schemas.android.com/apk/res/android"
android:shareInterpolator="false">
<rotate
android:fromDegrees="0"
android:toDegrees="360"
```

```
android:pivotX="50%"
android:pivotY="50%"
android:duration="5000" />
</set>
```

**Defining Tweened Animations Programmatically :**

You can programmatically define these animations.The different types of transformations are available as classes within the android.view.animation package. For example, you can define the aforementioned rotation animation as follows :

```
import android.view.animation.RotateAnimation;
...
RotateAnimation rotate = new RotateAnimation(
0, 360, RotateAnimation.RELATIVE_TO_SELF, 0.5f,
RotateAnimation.RELATIVE_TO_SELF, 0.5f);
rotate.setDuration(5000);
```

**Defining Simultaneous and Sequential Tweened Animations :**

Animation transformations can happen simultaneously or sequentially when you set the startOffset and duration properties, which control when and for how long an animation takes to complete. You can combine animations into the <set> tag (programmatically, using AnimationSet) to share properties.

For example, the following animation resource file /res/anim/grow.xml includes a set of two scale animations :

First,we take 2.5 seconds to double in size, and then at 2.5 seconds, we start a second animation to shrink back to our starting size :

```
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android=
http://schemas.android.com/apk/res/android
android:shareInterpolator="false">
<scale
android:pivotX="50%"
android:pivotY="50%"
android:fromXScale="1.0"
android:fromYScale="1.0"
android:toXScale="2.0"
android:toYScale="2.0"
android:duration="2500" />
<scale
android:startOffset="2500"
android:duration="2500"
android:pivotX="50%"
android:pivotY="50%"
android:fromXScale="1.0"
```

```
android:fromYScale="1.0"
android:toXScale="0.5"
android:toYScale="0.5" />
</set>
```

**Loading Animations :**

Loading animations is made simple by using the AnimationUtils helper class. The following code loads an animation XML resource file called /res/anim/grow.xml and applies it to an ImageView whose source resource is a green rectangle shape drawable :

```
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
...
ImageView iView =
(ImageView)findViewById(R.id.ImageView1);
iView.setImageResource(R.drawable.green_rect);
Animation an =
AnimationUtils.loadAnimation(this, R.anim.grow);
iView.startAnimation(an);
```

We can listen for Animation events, including the animation start, end, and repeat events, by implementing an AnimationListener class, such as the MyListener class shown here :

```
class MyListener implements Animation.
AnimationListener {
public void onAnimationEnd(Animation animation) {
// Do at end of animation
}
public void onAnimationRepeat(Animation animation) {
// Do each time the animation loops
}
public void onAnimationStart(Animation animation) {
// Do at start of animation
}
}
```

You can then register your AnimationListener as follows:

```
an.setAnimationListener(new MyListener());
```

**Exploring the Four Different Tweening Transformations :**

Now let's look at each of the four types of tweening transformations individually. These types are

- Transparency changes (Alpha)
- Rotations (Rotate)

- Scaling (Scale)
- Movement (Translate)

**Working with Alpha Transparency Transformations :**

Transparency is controlled using Alpha transformations. Alpha transformations can be used to fade objects in and out of view or to layer them on the screen.

Alpha values range from 0.0 (fully transparent or invisible) to 1.0 (fully opaque or visible).

Alpha animations involve a starting transparency (fromAlpha) and an ending transparency (toAlpha).

The following XML resource file excerpt defines a transparency–change animation, taking five seconds to fade in from fully transparent to fully opaque :

```
<alpha
android:fromAlpha="0.0"
android:toAlpha="1.0"
android:duration="5000">
</alpha>
```

Programmatically, you can create this same animation using the AlphaAnimation class within the android.view.animation package.

**Working with Rotating Transformations :**

You can use rotation operations to spin objects clockwise or counterclockwise around a pivot point within the object's boundaries.

Rotations are defined in terms of degrees. For example, you might want an object to make one complete clockwise rotation. To do this, you set the fromDegrees property to 0 and the toDegrees property to 360. To rotate the object counterclockwise instead, you set the toDegrees property to –360.

By default, the object pivots around the (0,0) coordinate, or the top–left corner of the object. This is great for rotations such as those of a clock's hands, but much of the time, you want to pivot from the center of the object; you can do this easily by setting the pivot point, which can be a fixed coordinate or a percentage.

The following XML resource file excerpt defines a rotation animation, taking five seconds to make one full clockwise rotation, pivoting from the center of the object :

```
<rotate
android:fromDegrees="0"
android:toDegrees="360"
android:pivotX="50%"
android:pivotY="50%"
android:duration="5000" />
```

Programmatically, you can create this same animation using the RotateAnimation class within the android.view.animation package.

**Working with Scaling Transformations :**

You can use scaling operations to stretch objects vertically and horizontally. Scaling operations are defined as relative scales. Think of the scale value of 1.0 as 100 percent, or fullsize.

To scale to half–size, or 50 percent, set the target scale value of 0.5.

You can scale horizontally and vertically on different scales or on the same scale (to preserve aspect ratio). You need to set four values for proper scaling: starting scale (fromXScale, fromYScale) and target scale (toXScale, toYScale). Again, you can use a pivot point to stretch your object from a specific (x,y) coordinate such as the center or another coordinate.

The following XML resource file excerpt defines a scaling animation, taking five seconds to double an object's size, pivoting from the center of the object :

```
<scale
android:pivotX="50%"
android:pivotY="50%"
android:fromXScale="1.0"
android:fromYScale="1.0"
android:toXScale="2.0"
android:toYScale="2.0"
android:duration="5000" />
```

Programmatically, you can create this same animation using the ScaleAnimation class within the android.view.animation package.

**Working with Moving Transformations :**

You can move objects around using translate operations.Translate operations move an object from one position on the (x,y) coordinate to another coordinate.

To perform a translate operation, you must specify the change, or delta, in the object's coordinates. You can set four values for translations: starting position (fromXDelta, fromYDelta) and relative target location (toXDelta, toYDelta).

The following XML resource file excerpt defines a translate animation, taking 5 seconds to move an object up (negative) by 100 on the y–axis. We also set the fillAfter. property to be true, so the object doesn't "jump" back to its starting position when the animation finishes:

```
<translate android:toYDelta="-100"
android:fillAfter="true"
android:duration="2500" />
```

Programmatically, you can create this same animation using the TranslateAnimation class within the android.view.animation package.

**Working with Different Interpolators :**

The animation interpolator determines the rate at which a transformation happens in time. There are a number of different interpolators provided as part of the Android SDK framework. Some of these interpolators include n AccelerateDecelerateInterpolator : Animation starts slowly, speeds up, and ends slowly

- AccelerateInterpolator : Animation starts slowly and then accelerates
- AnticipateInterpolator : Animation starts backward, and then flings forward
- AnticipateOvershootInterpolator : Animation starts backward, flings forward, overshoots its destination, and then settles at the destination
- BounceInterpolator : Animation "bounces" into place at its destination
- CycleInterpolator : Animation is repeated a certain number of times smoothly transitioning from one cycle to the next
- DecelerateInterpolator : Animation begins quickly, and then decelerates
- LinearInterpolator : Animation speed is constant throughout
- OvershootInterpolator : Animation overshoots its destination, and then settles at the destination

You can specify the interpolator used by an animation programmatically using the setInterpolator() method or in the animation XML resource using the android:interpolator attribute.

❑ **Check Your Progress :**

1. How many types of animation available in Android ?

   (A) One (B) Two (C) Three (D) Four

2. Which method is used to start the Animation ?

   (A) startAnim() (B) start()

   (C) startAnimation() (D) Animation()

3. Which method is used to load the Animation ?

   (A) Load() (B) loadAnimation()

   (C) load() (D) loadAnim()

4. Images and Shapes are stored in ________ folder.

   (A) asset (B) src (C) Layout (D) Drawable

5. Animation can be found in ________ package in android.

   (A) android.animation (B) android.layout

   (C) android.view (D) android.view.animation

## 10.4 Let Us Sum Up :

In this unit we discussed regarding to learn working with drawing & Animation, to learn & understand the use of Canvas and Paint, to understand the work with bitmap and shapes & to understand how to set and apply animation in Android Application

## 10.5 Answers for Check Your Progress :

**1.** (B) **2.** (C) **3.** (B) **4.** (D) **5.** (D)

## 10.6 Glossary :

1. **SDK :** Software Development Kit
2. **AVD :** Android Virtual Device
3. **DDMS :** Dalvik Debug Monitor Server
4. **ADB :** Android Debug Bridge

## 10.7 Assignment :

1. What is Animation ? Explain Frame–by–Frame and Tweened Animation in details.
2. Explain the Different twining Transformation

## 10.8 Activities :

1. Apply the various methods of amination in application and check the type of animation.

## 10.9 Case Study :

Create application in android which shows the use of both frame by frame and twined animation.

## 10.10 Further Reading :

Lauren Darcey and Shane Conder, "Android Wireless Application Development", Pearson Education, 2nd ed. (2011).

## BLOCK SUMMARY :

In this block we have seen so far regarding to understand the user interface screen elements, to design the user interface and layouts and to understand working with Drawing and Amination.

## BLOCK ASSIGNMENT :

1. What is Animation? Explain Frame–by–Frame and Twined Animation in details.
2. Explain the Different twining Transformation.
3. Explain different Layout available in Android.
4. How to retrieve data from user ? Discuss with example.
5. Explain different Common user interface elements available in Android.
6. What is menu ? Which two types of menu available in Android ?
7. For what purpose Chronometer is used ? Explain the Chronometer object's format attribute explain the different methods associated with Chronometer.
8. Explain Different Types of Dialogs with lifecycle in Android.
9. Which method and attribute is used to set image of ImageView ?

❖ **Short Questions :**

1. What is Animation ?
2. List out type of Layouts in Android.
3. Difference between frame by frame animation and twined animation.
4. How to load the animation ?
5. Which method is used to start the animation ?

❖ **Long Questions :**

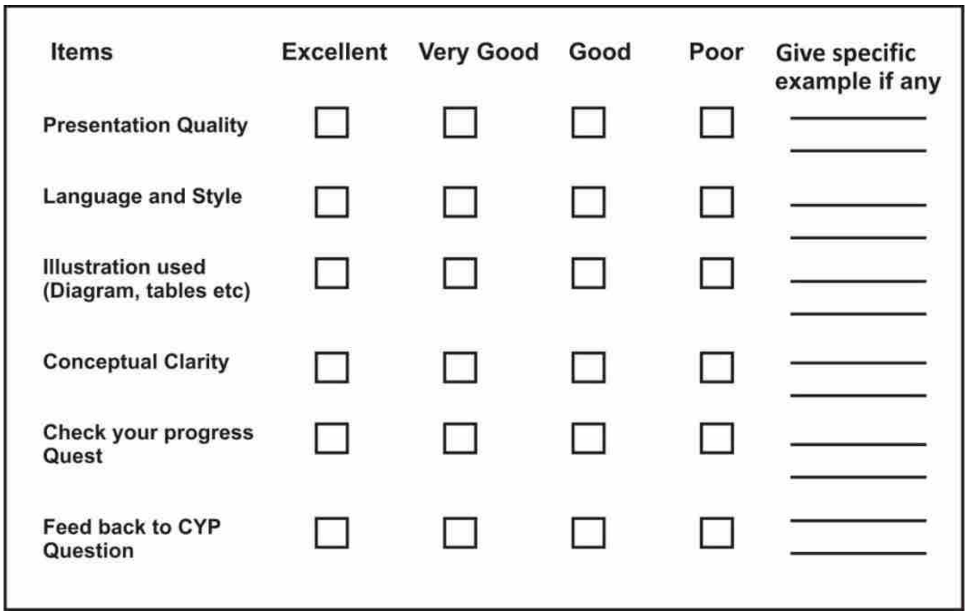
1. What is Animation ? Explain Frame–by–Frame and Twined Animation in details.
2. Explain the Different twining Transformation.
3. Explain different Layout available in Android.
4. How to retrieve data from user ? Discuss with example.
5. Explain different Common user interface elements available in Android.
6. What is menu ? Which two types of menu available in Android ?
7. For what purpose Chronometer is used ? Explain the Chronometer object's format attribute explain the different methods associated with Chronometer.
8. Explain Different Types of Dialogs with lifecycle in Android.
9. Which method and attribute is used to set image of ImageView ?

❖ **Enrolment No. :** ☐

1. How many hours did you need for studying the units ?

| Unit No. | 8 | 9 | 10 |
|---|---|---|---|
| No. of Hrs. | | | |

2. Please give your reactions to the following items based on your reading of the block :



| Items | Excellent | Very Good | Good | Poor | Give specific example if any |
|---|---|---|---|---|---|
| Presentation Quality | ☐ | ☐ | ☐ | ☐ | ________ |
| Language and Style | ☐ | ☐ | ☐ | ☐ | ________ |
| Illustration used (Diagram, tables etc) | ☐ | ☐ | ☐ | ☐ | ________ |
| Conceptual Clarity | ☐ | ☐ | ☐ | ☐ | ________ |
| Check your progress Quest | ☐ | ☐ | ☐ | ☐ | ________ |
| Feed back to CYP Question | ☐ | ☐ | ☐ | ☐ | ________ |

3. Any other Comments

..........................................................................................................

..........................................................................................................

..........................................................................................................

..........................................................................................................

..........................................................................................................

..........................................................................................................

..........................................................................................................

..........................................................................................................

**Dr. Babasaheb Ambedkar Open University Ahmedabad**

**BCAR-503**

# *MOBILE APPLICATION DEVELOPMENT (USING ANDROID)*

**BLOCK 4 : COMMON ANDROID APIS & DEPLOYING ANDROID APPLICATION**

# COMMON ANDROID APIS & DEPLOYING ANDROID APPLICATION

## Block Introduction :

Applications are about functionality and data. In this Block, we explore the various ways you can store, manage, and share application data with Android. Applications can store and manage data in different ways. For example, applications can use a combination of application preferences, the file system, and built–in SQLite database support to store information locally. The methods your application uses depend on your requirements. In this chapter, you learn how to use each of these mechanisms to store, retrieve, and interact with data.

## Block Objectives :

- To understand the common API available in Android.
- To implement Database Connection Application using SQLite Database.
- To understand the network and web connection using implementation of Network and Web APIs.
- To create the Android Application for Calling & Message sending using Android Telephony API.
- To understand how to deploy Android and sell the Android Application.

## Block Structure :

Unit 11

# MANAGING DATA USING SQLITE

## UNIT STRUCTURE

## 11.0 Learning Objectives :

- To learn how to use SQLite Database in Android Application
- To learn how to create Database Application using SQLite Database
- To understand and apply various Object and Methods available in SQLite database

## 11.1 Introduction :

For occasions when your application requires a more robust data storage mechanism, the Android file system includes support for application–specific relational databases using SQLite. SQLite databases are lightweight and file–based, making them ideally suited for embedded devices.

These databases and the data within them are private to the application. To share application data with other applications, you must expose the data you want to share by making your application a content provider (discussed later in this chapter).

The Android SDK includes a number of useful SQLite database management classes. Many of these classes are found in the *android.database.sqlite* package. Here you can find utility classes for managing database creation and versioning, database management, and query builder helper classes to help you format proper SQL statements and queries.

The package also includes specialized Cursor objects for iterating query results. You can also find all the specialized exceptions associated with SQLite.

Here we focus on creating databases within our Android applications. For that, we use the built–in SQLite support to programmatically create and use a SQLite database to store application information. However, if your application works with a different sort of database, you can also find more generic database classes (within the *android.database* package) to help you work with data from other providers. In addition to programmatically creating and using SQLite databases, developers can also interact directly with their application's database using the sqlite3 command–line tool that's accessible through the ADB shell interface. This can be an extremely helpful debugging tool for developers and quality assurance personnel, who might want to manage the database state (and content) for testing purposes.

## 11.2 Creating a SQLite Database :

You can create a SQLite database for your Android application in several ways. To illustrate how to create and use a simple SQLite database, let's create an Android project called Simple Database.

### 11.2.1 Creating a SQLite Database Instance Using the Application Context :

The simplest way to create a new *SQLiteDatabase* instance for your application is to use the *openOrCreateDatabase()* method of your application Context, like this :

```
import android.database.sqlite.SQLiteDatabase;
...
SQLiteDatabase mDatabase;
mDatabase =
openOrCreateDatabase("my_sqlite_database.db",
SQLiteDatabase.CREATE_IF_NECESSARY, null);
```

### 11.2.2 Finding the Application's Database File on the Device File System :

Android applications store their databases (SQLite or otherwise) in a special application directory: */data/data/<application package name>/databases/ <databasename>*

So, in this case, the path to the database would be

*/data/data/com.androidbook.SimpleDatabase/databases/my_sqlite_ database.db*

You can access your database using the sqlite3 command–line interface using this path.

### 11.2.3 Configuring the SQLite Database Properties :

Now that you have a valid SQLiteDatabase instance, it's time to configure it. Some important database configuration options include version, locale, and the thread–safe locking feature.

```
import java.util.Locale;
...
mDatabase.setLocale(Locale.getDefault());
mDatabase.setLockingEnabled(true);
    mDatabase.setVersion(1);
```

### 11.2.4 Creating Tables and Other SQLite Schema Objects :

Creating tables and other SQLite schema objects is as simple as forming proper SQLite statements and executing them. The following is a valid CREATE TABLE SQL statement. This statement creates a table called tbl_authors.The table has three fields: a unique id number, which auto–increments with each record and acts as our primary key, and firstname and lastname text fields :

```
CREATE TABLE tbl_authors (
id INTEGER PRIMARY KEY AUTOINCREMENT,
firstname TEXT,
lastname TEXT);
```

You can encapsulate this CREATE TABLE SQL statement in a static final String variable (called CREATE_AUTHOR_TABLE) and then execute it on your database using the execSQL() method :

*mDatabase.execSQL(CREATE_AUTHOR_TABLE);*

The execSQL() method works for nonqueries.You can use it to execute any valid SQLite SQL statement. For example, you can use it to create, update, and delete tables, views, triggers, and other common SQL objects. In our application,we add another table called tbl_books. The schema for tbl_books looks like this :

```
CREATE TABLE tbl_books (
id INTEGER PRIMARY KEY AUTOINCREMENT,
title TEXT,
dateadded DATE,
authorid INTEGER NOT NULL CONSTRAINT authorid
    REFERENCES tbl_authors(id) ON DELETE
CASCADE);
```

Unfortunately, SQLite does not enforce foreign key constraints. Instead, we must enforce them ourselves using custom SQL triggers. So we create triggers, such as this one that enforces that books have valid authors :

```
private static final String CREATE_TRIGGER_ADD =
"CREATE TRIGGER fk_insert_book BEFORE INSERT ON
tbl_books
FOR EACH ROW
    BEGIN
SELECT RAISE(ROLLBACK, 'insert on table \"tbl_books\"
    violates foreign key
constraint \"fk_authorid\"') WHERE (SELECT id FROM
    tbl_authors WHERE id =
NEW.authorid) IS NULL;
END;";
```

We can then create the trigger simply by executing the CREATE TRIGGER SQL statement :

```
mDatabase.execSQL(CREATE_TRIGGER_ADD);
```

We need to add several more triggers to help enforce our link between the author and book tables, one for updating *tbl_books* and one for deleting records from *tbl_authors*.

## 11.3 Creating, Updating, and Deleting Database Records :

Now that we have a database set up, we need to create some data. The SQLiteDatabase class includes three convenience methods to do that. They are, as you might expect, insert(), update(), and delete().

### 11.3.1 Inserting Records :

We use the *insert()* method to add new data to our tables. We use the *ContentValues* object to pair the column names to the column values for the record we want to insert.

For example, here we insert a record into *tbl_authors* for J.K. Rowling :

```
import android.content.ContentValues;
...
ContentValues values = new ContentValues();
values.put("firstname", "J.K.");
values.put("lastname", "Rowling");
```

```
long newAuthorID = mDatabase.insert("tbl_authors",
null, values);
```

The *insert()* method returns the id of the newly created record. We use this author id to create book records for this author.

You might want to create simple classes (that is, class Author and class Book) to encapsulate your application record data when it is used programmatically.

### 11.3.2 Updating Records :

You can modify records in the database using the *update()* method. The update() method takes four arguments :

- The table to update records
- A *ContentValues* object with the modified fields to update
- An optional WHERE clause, in which ? identifies a WHERE clause argument
- An array of WHERE clause arguments, each of which is substituted in place of the ?'s from the second parameter Passing null to the WHERE clause modifies all records within the table, which can be useful for making sweeping changes to your database.

Most of the time, we want to modify individual records by their unique identifier.

The following function takes two parameters : an updated book title and a *bookId*. We find the record in the table called *tbl_books* that corresponds with the id and update that book's title. Again, we use the *ContentValues* object to bind our column names to our data values :

```
public void updateBookTitle(Integer bookId, String
newtitle) {
ContentValues values = new ContentValues();
values.put("title", newtitle);
mDatabase.update("tbl_books",
values, "id=?", new String[] { bookId.toString() });
}
```

Because we are not updating the other fields, we do not need to include them in the *ContentValues* object. We include only the title field because it is the only field we change.

### 11.3.3 Deleting Records :

You can remove records from the database using the *remove()* method. The remove() method takes three arguments:

- The table to delete the record from
- An optional WHERE clause, in which ? identifies a WHERE clause argument
- An array of WHERE clause arguments, each of which is substituted in place of the ?'s from the second parameter

Passing null to the WHERE clause deletes all records within the table. For example, this function call deletes all records within the table called tbl_authors :

```
mDatabase.delete("tbl_authors", null, null);
```

Most of the time, though, we want to delete individual records by their unique identifiers.

The following function takes a parameter *bookId* and deletes the record corresponding to that unique id (primary key) within the table called *tbl_books* :

```
public void deleteBook(Integer bookId) {
mDatabase.delete("tbl_books", "id=?",
new String[] { bookId.toString() });
}
```

You need not use the primary key (id) to delete records; the WHERE clause is entirely up to you. For instance, the following function deletes all book records in the table *tbl_books* for a given author by the author's unique id :

```
public void deleteBooksByAuthor(Integer authorID) {
int numBooksDeleted = mDatabase.delete("tbl_books",
"authorid=?",
new String[] { authorID.toString() });
}
```

**11.3.4 Working with Transactions :**

Often you have multiple database operations you want to happen all together or not at all.

You can use SQL Transactions to group operations together; if any of the operations fails, you can handle the error and either recover or roll back all operations. If the operations all succeed, you can then commit them. Here we have the basic structure for a transaction :

```
mDatabase.beginTransaction();
try {
// Insert some records, updated others, delete a few
// Do whatever you need to do as a unit, then commit
it
mDatabase.setTransactionSuccessful();
} catch (Exception e) {
// Transaction failed. Failed! Do something here.
// It's up to you.
} finally {
mDatabase.endTransaction();
}
```

Now let's look at the transaction in a bit more detail. A transaction always begins with a call to *beginTransaction()* method and a try/catch block. If your operations are successful, you can commit your changes with a call to the *setTransactionSuccessful()* method. If you do not call this method, all your operations are rolled back and not committed.

Finally, you end your transaction by calling *endTransaction().* It's as simple as that.

In some cases, you might recover from an exception and continue with the transaction.

For example, if you have an exception for a read–only database, you can open the database and retry your operations.

Finally, note that transactions can be nested, with the outer transaction either committing or rolling back all inner transactions.

## 11.4 Querying SQLite Databases :

Databases are great for storing data in any number of ways, but retrieving the data you want is what makes databases powerful. This is partly a matter of designing an appropriate database schema, and partly achieved by crafting SQL queries, most of which are SELECT statements.

Android provides many ways in which you can query your application database. You can run raw SQL query statements (strings), use a number of different SQL statement builder utility classes to generate proper query statements from the ground up, and bind specific user interface controls such as container views to your backend database directly.

### 11.4.1 Working with Cursors :

When results are returned from a SQL query, you often access them using a Cursor found in the *android.database.Cursor* class. Cursor objects are rather like file pointers; they allow random access to query results.

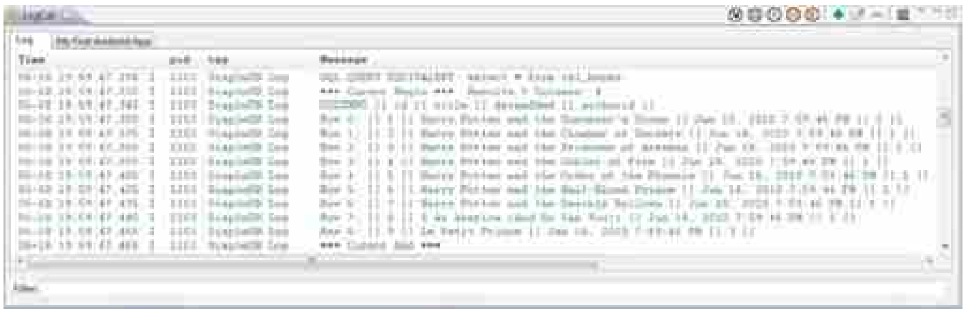
You can think of query results as a table, in which each row corresponds to a returned record. The Cursor object includes helpful methods for determining how many results were returned by the query the Cursor represents and methods for determining the column names (fields) for each returned record. The columns in the query results are defined by the query, not necessarily by the database columns. These might include calculated columns, column aliases, and composite columns.

Cursor objects are generally kept around for a time. If you do something simple (such as get a count of records or in cases when you know you retrieved only a single simple record), you can execute your query and quickly extract what you need; don't forget to close the Cursor when you're done, as shown here :

```
// SIMPLE QUERY: select * from tbl_books
Cursor c = mDatabase.query
("tbl_books",null,null,null,null,null,null);
// Do something quick with the Cursor here...
c.close();
```

### 11.4.2 Executing Simple Queries :

Your first stop for database queries should be the *query()* methods available in the *SQLiteDatabase* class. This method queries the database and returns any results as in a Cursor object



```
[illegible]
```

**Figure 11.1 : Sample log output for the logCursorInfo() method.**

The query() method we mainly use takes the following parameters:

- [String] : The name of the table to compile the query against
- [String Array] : List of specific column names to return (use null for all)
- [String] The WHERE clause : Use null for all; might include selection args as ?'s
- [String Array] : Any selection argument values to substitute in for the ?'s in the earlier parameter
- [String] GROUP BY clause : null for no grouping
- [String] HAVING clause : null unless GROUP BY clause requires one
- [String] ORDER BY clause : If null, default ordering used
- [String] LIMIT clause : If null, no limit

Previously in the chapter, we called the query() method with only one parameter set to the table name.

```
Cursor c = mDatabase.query("tbl_books",null,null,null,
null,null,null);
```

This is equivalent to the SQL query

```
SELECT * FROM tbl_books;
```

Add a WHERE clause to your query, so you can retrieve one record at a time :

```
Cursor c = mDatabase.query("tbl_books", null,
"id=?", new String[]{"9"}, null, null, null);
This is equivalent to the SQL query
SELECT * tbl_books WHERE id=9;
```

Selecting all results might be fine for tiny databases, but it is not terribly efficient. You should always tailor your SQL queries to return only the results you require with no extraneous information included. Use the powerful language of SQL to do the heavy lifting.

for you whenever possible, instead of programmatically processing results yourself. For example, if you need only the titles of each book in the book table, you might use the following call to the query() method :

```
String asColumnsToReturn[] = { "title", "id" };
String strSortOrder = "title ASC";
Cursor c = mDatabase.query("tbl_books",
asColumnsToReturn,
null, null, null, null, strSortOrder);
```

This is equivalent to the SQL query

```
SELECT title, id FROM tbl_books ORDER BY title ASC;
```

## 11.5 Closing and Deleting a SQLite Database :

Although you should always close a database when you are not using it, you might on occasion also want to modify and delete tables and delete your database.

### 11.5.1 Deleting Tables and Other SQLite Objects :

You delete tables and other SQLite objects in exactly the same way you create them. Format the appropriate SQLite statements and execute them. For example, to drop our tables and triggers, we can execute three SQL statements :

```
mDatabase.execSQL("DROP TABLE tbl_books;");
mDatabase.execSQL("DROP TABLE tbl_authors;");
mDatabase.execSQL("DROP TRIGGER IF EXISTS
fk_insert_book;");
```

### 11.5.2 Closing a SQLite Database :

You should close your database when you are not using it. You can close the database using the *close()* method of your *SQLiteDatabase* instance, like this :

```
mDatabase.close();
```

### 11.5.3 Deleting a SQLite Database Instance Using the Application Context :

The simplest way to delete a *SQLiteDatabase* is to use the d*eleteDatabase()* method of your application Context. You delete databases by name and the deletion is permanent.

You lose all data and schema information.

```
deleteDatabase("my_sqlite_database.db");
```

## 11.6 Designing Persistent Databases :

Generally speaking, an application creates a database and uses it for the rest of the application's lifetime–by which we mean until the application is uninstalled from the phone. So far, we've talked about the basics of creating a database, using it, and then deleting it.

In reality, most mobile applications do not create a database on–the–fly, use them, and then delete them. Instead, they create a database the first time they need it and then use it.

The Android SDK provides a helper class called *SQLiteOpenHelper* to help you manage your application's database.

To create a SQLite database for your Android application using the *SQLiteOpenHelper*, you need to extend that class and then instantiate an instance of it as a member variable for use within your application. To illustrate how to do this, let's create a new Android project called *PetTracker*.

### 11.6.1 Keeping Track of Database Field Names :

You've probably realized by now that it is time to start organizing your database fields programmatically to avoid typos and such in your SQL queries. One easy way you do this is to make a class to encapsulate your database schema in a class, such as *PetDatabase*, shown here :

```
import android.provider.BaseColumns;
public final class PetDatabase {
private PetDatabase() {}
public static final class Pets implements BaseColumns {
private Pets() {}
public static final String
PETS_TABLE_NAME="table_pets";
public static final String PET_NAME="pet_name";
public static final String PET_TYPE_ID="pet_type_id";
public static final String DEFAULT_SORT_ORDER=
    "pet_name ASC";
}
public static final class PetType implements
BaseColumns {
private PetType() {}
public static final String PETTYPE_TABLE_NAME=
    "table_pettypes";
public static final String PET_TYPE_NAME="pet_type";
public static final String DEFAULT_SORT_ORDER=
    "pet_type ASC";
}
}
```

By implementing the *BaseColumns* interface, we begin to set up the underpinnings for using database–friendly user interface controls in the future, which often require a specially named column called _id to function properly. We rely on this column as our primary key.

**11.6.2 Extending the SQLiteOpenHelper Class :**

To extend the SQLiteOpenHelper class,we must implement several important methods, which help manage the database versioning.The methods to override are onCreate(), onUpgrade(), and onOpen().We use our newly defined PetDatabase class to generate appropriate SQL statements, as shown here:

```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import com.androidbook.PetTracker.PetDatabase.
    PetType;
import com.androidbook.PetTracker.PetDatabase.Pets;
class PetTrackerDatabaseHelper extends
    SQLiteOpenHelper {
private static final String DATABASE_NAME =
    "pet_tracker.db";
private static final int DATABASE_VERSION = 1;
PetTrackerDatabaseHelper(Context context) {
super(context, DATABASE_NAME, null, DATABASE_VERSION);
}
@Override
public void onCreate(SQLiteDatabase db) {
db.execSQL("CREATE TABLE " +PetType.PETTYPE_TABLE_
    NAME+" (" + PetType._ID + " INTEGER PRIMARY KEY
    AUTOINCREMENT ," + PetType.PET_TYPE_NAME + "
    TEXT" + ");");
db.execSQL("CREATE TABLE " + Pets.PETS_TABLE_
    NAME + " (" + Pets._ID + " INTEGER PRIMARY KEY
    AUTOINCREMENT ," + Pets.PET_NAME + " TEXT,"
    + Pets.PET_TYPE_ID + " INTEGER" // FK to pet type
    table + ");");
}
@Override
public void onUpgrade(SQLiteDatabase db, int
    oldVersion, int newVersion){
// Housekeeping here.
// Implement how "move" your application data
// during an upgrade of schema versions
// Move or delete data as required. Your call.
}
@Override
public void onOpen(SQLiteDatabase db) {
super.onOpen(db);
}
    }
```

Now we can create a member variable for our database like this:

```
PetTrackerDatabaseHelper mDatabase = new
PetTrackerDatabaseHelper(this.getApplicationContext());
```

Now, whenever our application needs to interact with its database, we request a valid database object. We can request a read–only database or a database that we can also write to. We can also close the database. For example, here we get a database we can write data to:

```
SQLiteDatabase db = mDatabase.getWritableDatabase();
```

## 11.7 Binding Data to the Application User Interface :

In many cases with application databases, you want to couple your user interface with the data in your database. You might want to fill drop–down lists with values from a database table, or fill out form values, or display only certain results. There are various ways to bind database data to your user interface. You, as the developer, can decide whether to use built in data–binding functionality provided with certain user interface controls, or you can build your own user interfaces from the ground up.

### 11.7.1 Working with Database Data Like Any Other Data :

If you peruse the *PetTracker* application provided on the book website, you notice that its functionality includes no magical data–binding features, yet the application clearly uses the database as part of the user interface.

Specifically, the database is leveraged :

- When you fill out the Pet Type field, the AutoComplete feature is seeded with pet types already in listed in the table_pettypes table (Figure 11.2, left).

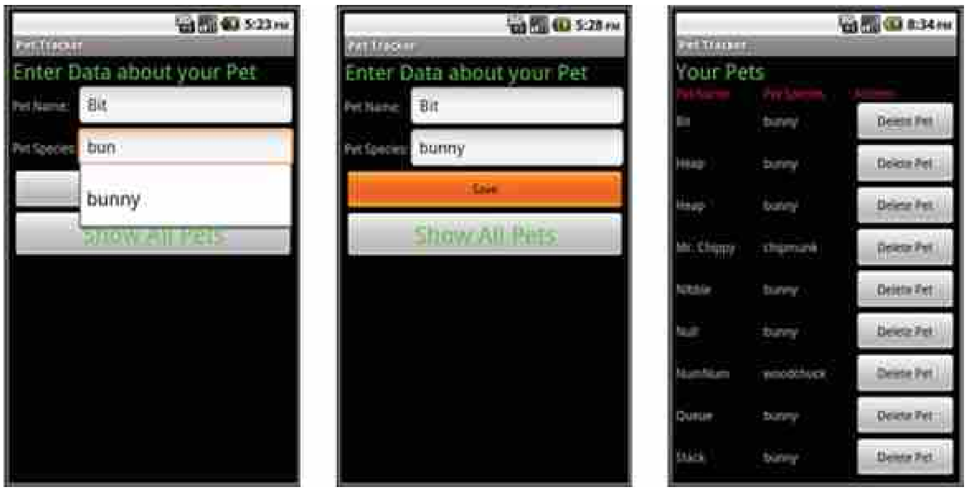


**Figure 11.2 : The PetTracker application :**
**Entry Screen (left, middle) and Pet Listing Screen (right).**

- When you save new records using the Pet Entry Form (Figure 11.2, middle).
- When you display the Pet List screen, you query for all pets and use a Cursor to programmatically build a *TableLayout* on–the–fly (Figure 11.2, right).

This might work for small amounts of data; however, there are various drawbacks to this method. For example, all the work is done on the main thread, so the more records you add, the slower your application response time becomes. Second, there's quite a bit of custom code involved to map the database results to the individual user interface components. If you decide you want to use a different control to display your data, you have quite a lot of reworks to do. Third, we constantly requery the database for fresh results, and we might be re querying far more than necessary.

### 11.7.2 Binding Data to Controls Using Data Adapters :

Ideally, you'd like to bind your data to user interface controls and let them take care of the data display. For example, we can use a fancy *ListView* to display the pets instead of building a *TableLayout* from scratch. We can spin through our Cursor and generate *ListView* child items manually, or even better, we can simply create a data adapter to map the Cursor results to each *TextView* child within the *ListView*. We included a project called PetTracker2 on the book website that does this. It behaves much like the PetTracker sample application, except that it uses the *SimpleCursorAdapter* with *ListView* and an *ArrayAdapter* to handle *AutoCompleteTextView* features.

**Binding Data Using SimpleCursorAdapter :**

Let's now look at how we can create a data adapter to mimic our Pet Listing screen, with each pet's name and species listed. We also want to continue to have the ability to delete records from the list.

Remember from Chapter 8, "Designing User Interfaces with Layouts," that the ListView container can contain children such as TextView objects. In this case, we want to display each Pet's name and type.We therefore create a layout file called pet_item.xml that becomes our ListView item template:

```
<?xml version="1.0" encoding="utf–8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/RelativeLayoutHeader"
android:layout_height="wrap_content"
android:layout_width="fill_parent">
<TextView
android:id="@+id/TextView_PetName"
android:layout_width="wrap_content"
android:layout_height="?android:attr/listPreferredItemHeight"
android:layout_alignParentLeft="true" />
<TextView
android:id="@+id/TextView_PetType"
android:layout_width="wrap_content"
android:layout_height="?android:attr/listPreferredItemHeight"
android:layout_alignParentRight="true" />
</RelativeLayout>
```

Next, in our main layout file for the Pet List, we place our ListView in the appropriate place on the overall screen. The ListView portion of the layout file might look something like this :

```
<ListView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/petList" android:divider="#000" />
```

Now to programmatically fill our ListView,we must take the following steps :

1. Perform our query and return a valid Cursor (a member variable).
2. Create a data adapter that maps the Cursor columns to the appropriate TextView controls within our pet_item.xml layout template.
3. Attach the adapter to the ListView.

In the following code,we perform these steps:

```
SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();
queryBuilder.setTables(Pets.PETS_TABLE_NAME +", " +
PetType.PETTYPE_TABLE_NAME);
```

```
queryBuilder.appendWhere(Pets.PETS_TABLE_NAME + "." +
Pets.PET_TYPE_ID + "=" + PetType.PETTYPE_TABLE_NAME + "." +
PetType._ID);
String asColumnsToReturn[] = { Pets.PETS_TABLE_NAME + "." +
Pets.PET_NAME, Pets.PETS_TABLE_NAME +
"." + Pets._ID, PetType.PETTYPE_TABLE_NAME + "." +
PetType.PET_TYPE_NAME };
mCursor = queryBuilder.query(mDB, asColumnsToReturn, null, null,
null, null, Pets.DEFAULT_SORT_ORDER);
startManagingCursor(mCursor);
ListAdapter adapter = new SimpleCursorAdapter(this,
R.layout.pet_item, mCursor,
new String[]{Pets.PET_NAME, PetType.PET_TYPE_NAME},
new int[]{R.id.TextView_PetName, R.id.TextView_PetType });
ListView av = (ListView)findViewById(R.id.petList);
av.setAdapter(adapter);
```

Notice that the _id column as well as the expected name and type columns appears in the query. This is required for the adapter and ListView to work properly.

Using a ListView (Figure 11.3, left) instead of a custom user interface enables us to take advantage of the ListView control's built–in features, such as scrolling when the list becomes longer, and the ability to provide context menus as needed.The _id column is used as the unique identifier for each ListView child node. If we choose a specific item on the list,we can act on it using this identifier, for example, to delete the item.

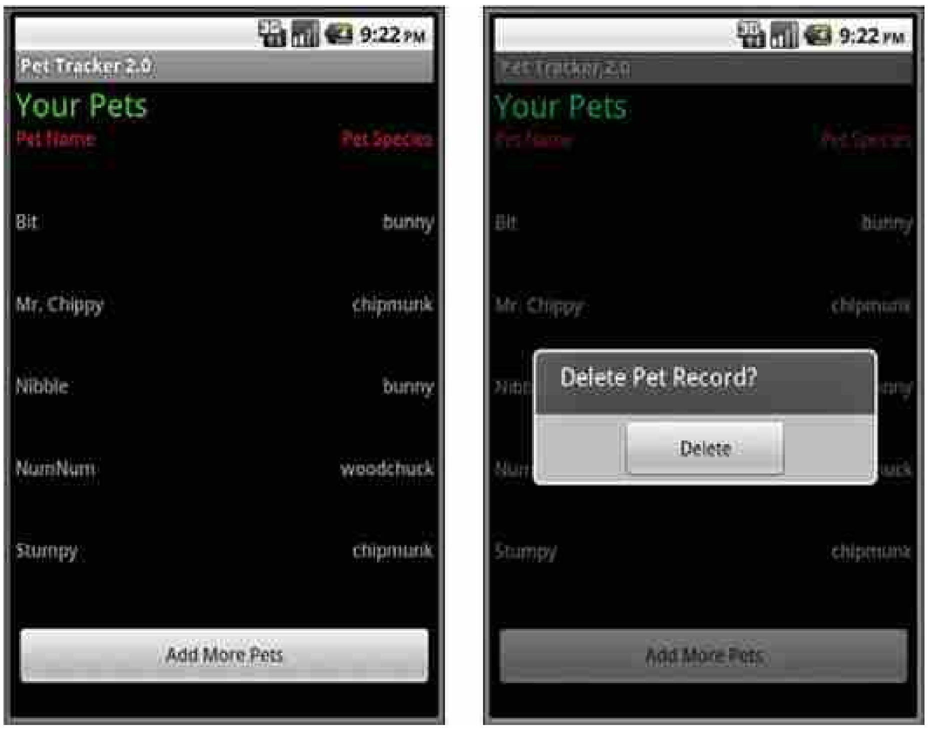


**Figure 11.3 : The PetTracker2 application : Pet Listing Screen ListView (left) with Delete feature (right).**

Now we re–implement the Delete functionality by listening for onItemClick() events and providing a Delete Confirmation dialog (Figure 11.3, right) :

```
av.setOnItemClickListener(new AdapterView.OnItemClickListener() {
public void onItemClick( AdapterView<?> parent, View view,
int position, long id) {
final long deletePetId = id;
new AlertDialog.Builder(PetTrackerListActivity.this).setMessage(
"Delete Pet Record?").setPositiveButton(
"Delete", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog,int which) {
deletePet(deletePetId);
mCursor.requery();
}}).show();
}
});
```

You can see what this would look like on the screen in Figure 11.3.

Note that within the PetTracker2 sample application,we also use an ArrayAdapter to bind the data in the pet_types table to the AutoCompleteTextView on the Pet Entry screen. Although our next example shows you how to do this in a preferred manner, we left this code in the PetTracker sample to show you that you can always intercept the data your Cursor provides and do what you want with it. In this case,we create a String array for the AutoText options by hand.We use a built–in Android layout resource called android.R.layout.simple_dropdown_item_1line to specify what each individual item within the AutoText listing looks like.You can find the built–in layout resources provided within your appropriate Android SDK version's resource subdirectory.

Storing Nonprimitive Types (Such as Images) in the Database

Because SQLite is a single file, it makes little sense to try to store binary data within the database. Instead store the location of data, as a file path or a URI in the database, and access it appropriately. We show an example of storing image URIs in the database in the next chapter.

## ❑ Check Your Progress :

1. SQLite supports many of the features of SQL and is fast; however, ________ procedures are not supported.

   (A) Function (B) Triger

   (C) View (D) Store Procedures

2. ________ databases can be accessed simultaneously with SQLite in the same session.

   (A) Single (B) Multiple

   (C) Triple (D) None of the Above

3. SQLite is available on ________ Operation System.
   (A) Android (B) MAC (C) Linux (D) Windows
4. How many types of SQLite commands are there ?
   (A) 2 (B) 3 (C) 4 (D) 5
5. Based on case–sensitivity, SQLite is ________
   (A) Case Sensitive
   (B) Not Case Sensitive
   (C) Not case sensitive with few commands being case sensitive
   (D) None of the Above

## 11.8 Let Us Sum Up :

In this unit we discussed regarding the various classes and methods available in android to implement data driven application by using the SQLite database.

## 11.9 Answers for Check Your Progress :

**1.** (D) **2.** (B) **3.** (A) **4.** (B) **5.** (C)

## 11.10 Glossary :

**1.** **SDK :** Software Development Kit
**2.** **AVD :** Android Virtual Device
**3.** **DDMS :** Dalvik Debug Monitor Server
**4.** **ADB :** Android Debug Bridge

## 11.11 Assignment :

1. Explain how to Create, Update, and Delete Database Records in SQLite with example.
2. What is Cursor ? Explain the use of Cursor in SQLite Database.
3. Explain the use of ContentValues to insert the record and Which four arguments are taken by update() ?
4. What is SQLite ?
5. How to create SQLite database ?
6. How to create tables in SQLite database ?

## 11.12 Activities :

1. Define the application which will have the record which can be processed and managed by the SQLite database with proper class and methods available in android.

## 11.13 Case Study :

Create android application which shows the data management by the SQLite database.

## 11.14 Further Reading :

Lauren Darcey and Shane Conder, "Android Wireless Application Development", Pearson Education, 2nd ed. (2011).

Unit 12

# USING ANDROID NETWORKING APIS

## UNIT STRUCTURE

## 12.0 Learning Objectives :

- To learn how to use Network API in Android Application
- To learn how to Access the Internet (HTTP)
- To understand how to Read Data from the Web
- To learn parsing the XML Data
- To understand how to get Network status

## 12.1 Introduction :

Applications written with networking components are far more dynamic and content rich than those that are not. Applications leverage the network for a variety of reasons: to deliver fresh and updated content, to enable social networking features of an otherwise standalone application, to offload heavy processing to high–powered servers, and to enable data storage beyond what the user can achieve on the device. Those accustomed to Java networking will find the java.net package familiar. There are also some helpful Android utility classes for various types of network operations and protocols. This chapter focuses on Hypertext Transfer Protocol (HTTP), the most common protocol for networked mobile applications.

Networking on the Android platform is standardized, using a combination of powerful yet familiar technologies and libraries such as java.net. Network implementation is generally straightforward, but mobile application developers need to plan for less stable connectivity than one might expect in a home or office network setting–connectivity depends on the location of the users and their devices. Users demand stable, responsive applications. This means that you must take extra care when designing network–enabled applications. Luckily, the Android SDK provides a number of tools and classes for ensuring just that.

## 12.2 Accessing the Internet (HTTP) :

The most common way to transfer data to and from the network is to use HTTP. You can use HTTP to encapsulate almost any type of data and to secure the data with Secure Sockets Layer (SSL), which can be important when you transmit data that falls under privacy requirements. Also, most common ports used by HTTP are typically open from the phone networks.

## 12.3 Reading Data from the Web :

Reading data from the Web can be extremely simple. For example, if all you need to do is read some data from a website and you have the web address of that data, you can leverage the URL class (available as part of the java.net package) to read a fixed amount of text from a file on a web server, like this :

```
import java.io.InputStream;
import java.net.URL;
// ...
URL text = new URL(
"http://api.flickr.com/services/feeds/photos_
public.gne" + "?id=26648248@N04&lang=en-us&format=
atom");
InputStream isText = text.openStream();
byte[] bText = new byte[250];
int readSize = isText.read(bText);
Log.i("Net", "readSize = " + readSize);
Log.i("Net", "bText = "+ new String(bText));
isText.close();
```

First, a new URL object is created with the URL to the data we want to read. A stream is then opened to the URL resource. From there,we read the data and close the InputStream. Reading data from a server can be that simple.

However, remember that because we work with a network resource, errors can be more common. Our phone might not have network coverage; the server might be down for maintenance or disappear entirely; the URL might be invalid; and network users might experience long waits and timeouts.

This method might work in some instances–for example, when your application has lightweight, noncritical network features–but it's not particularly elegant. In many cases, you might want to know more about the data before reading from it from the URL. For instance, you might want to know how big it is.

Finally, for networking to work in any Android application, permission is required.Your application needs to have the following statement in its AndroidManifest.xml file :

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

## 12.4 Using HttpURLConnection :

We can use the HttpURLConnection object to do a little reconnaissance on our URL before we transfer too much data. HttpURLConnection retrieves some information about the resource referenced by the URL object, including HTTP status and header information.

Some of the information you can retrieve from the HttpURLConnection includes the length of the content, content type, and date–time information so that you can check to see if the data changed since the last time you accessed the URL.

Here is a short example of how to use HttpURLConnection to query the same URL previously used :

```
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
// ...
URL text = new URL(
"http://api.flickr.com/services/feeds/
photos_public.gne
➥?id=26648248@N04&lang=en-us&format=atom");
HttpURLConnection http =
(HttpURLConnection)text.openConnection();
Log.i("Net", "length = " + http.getContentLength());
Log.i("Net", "respCode = " + http.getResponseCode());
Log.i("Net", "contentType = "+ http.getContentType());
Log.i("Net", "content = "+http.getContent());
```

The log lines demonstrate a few useful methods with the HttpURLConnection class. If the URL content is deemed appropriate, you can then call http.getInputStream() to get the same InputStream object as before. From there, reading from the network resource is the same, but more is known about the resource.

## 12.5 Parsing XML from the Network :

A large portion of data transmitted between network resources is stored in a structured fashion in Extensible Markup Language (XML). In particular, RSS feeds are provided in a standardized XML format, and many web services provide data using these feeds.

Android SDK provides a variety of XML utilities. We dabble with the XML Pull Parser in Chapter 6, "Managing Application Resources." We also cover the various SAX and DOM support available in Chapter 10, "Using Android Data and Storage APIs."

Parsing XML from the network is similar to parsing an XML resource file or a raw file on the file system. Android provides a fast and efficient XML Pull Parser, which is a parser of choice for networked applications.

The following code demonstrates how to use the XML Pull Parser to read an XML file from flickr.com and extract specific data from within it. A TextView called status is assigned before this block of code is executed and displays the status of the parsing operation.

```
import java.net.URL;
import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserFactory;
// ...
URL text = new URL(
"http://api.flickr.com/services/feeds/photos_
public.gne
➥?id=26648248@N04&lang=en-us&format=atom");
XmlPullParserFactory parserCreator =
XmlPullParserFactory.newInstance();
XmlPullParser parser = parserCreator.newPullParser();
parser.setInput(text.openStream(), null);
status.setText("Parsing...");
int parserEvent = parser.getEventType();
while (parserEvent != XmlPullParser.END_DOCUMENT) {
switch(parserEvent) {
case XmlPullParser.START_TAG:
String tag = parser.getName();
if (tag.compareTo("link") == 0) {
String relType =
parser.getAttributeValue(null, "rel");
if (relType.compareTo("enclosure") == 0 ) {
String encType =
    parser.getAttributeValue(null, "type");
if (encType.startsWith("image/")) {
String imageSrc =
parser.getAttributeValue(null, "href");
Log.i("Net",
"image source = " + imageSrc);
}
}
}
break;
}
parserEvent = parser.next();
}
status.setText("Done...");
```

After the URL is created, the next step is to retrieve an XmlPullParser instance from the XmlPullParserFactory. A Pull Parser has a main method that returns the next event. The events returned by a Pull Parser are similar to methods used in the implementation of a SAX parser handler class. Instead, though, the code is handled iteratively. This method is more efficient for mobile use.

In this example, the only event that we check for is the START_TAG event, signifying the beginning of an XML tag. Attribute values are queried and compared. This example looks specifically for image URLs within the XML from a flickr feed query.When found, a log entry is made.

You can check for the following XML Pull Parser events :

- **START_TAG :** Returned when a new tag is found (that is, <tag>)
- **TEXT :** Returned when text is found (that is, <tag>text</tag> where text has been found)
- **END_TAG :** Returned when the end of tag is found (that is, </tag>)
- **END_DOCUMENT :** Returned when the end of the XML file is reached

Additionally, the parser can be set to validate the input. Typically, parsing without validation is used when under constrained memory environments, such as a mobile environment.

Compliant, nonvalidating parsing is the default for this XML Pull Parser.

## 12.6 Processing Asynchronously :

Users demand responsive applications, so time–intensive operations such as networking should not block the main UI thread. The style of networking presented so far causes the UI thread it runs on to block until the operation finishes. For small tasks, this might be acceptable. However, when timeouts, large amounts of data, or additional processing, such as parsing XML, is added into the mix, you should move these time–intensive operations off of the main UI thread.

Offloading intensive operations such as networking provides a smoother, more stable experience to the user. The Android SDK provides two easy ways to manage offload processing from the main UI thread: the AsyncTask class and the standard Java Thread class. The AsyncTask class is a special class for Android development that encapsulates background processing and helps facilitate communication to the UI thread while managing the lifecycle of the background task within the context of the activity lifecycle. Developers can also construct their own threading solutions using the standard Java methods and classes–but they are then responsible for managing the entire thread lifecycle as well.

## 12.7 Working with AsyncTask :

AsyncTask is an abstract helper class for managing background operations that eventually post back to the UI thread. It creates a simpler interface for asynchronous operations than manually creating a Java Thread class.

Instead of creating threads for background processing and using messages and message handlers for updating the UI, you can create a subclass of *AsyncTask* and implement the appropriate event methods. The *onPreExecute()* method runs on the UI thread before background processing begins. The *doInBackground()* method handles background processing, whereas *publishProgress()* informs the UI thread periodically about the background processing progress. When the background processing finishes, the *onPostExecute()* method runs on the UI thread to give a final update.

The following code demonstrates an example implementation of *AsyncTask* to perform the same functionality as the code for the Thread :

```
private class ImageLoader extends
AsyncTask<URL, String, String> {
@Override
protected String doInBackground(
URL... params) {
// just one param
try {
URL text = params[0];
// ... parsing code {
publishProgress(
"imgCount = " + curImageCount);
// ... end parsing code }
}
catch (Exception e ) {
Log.e("Net",
"Failed in parsing XML", e);
    return "Finished with failure.";
}
return "Done...";
}
protected void onCancelled() {
Log.e("Net", "Async task Cancelled");
}
protected void onPostExecute(String result) {
mStatus.setText(result);
}
protected void onPreExecute() {
```

```
mStatus.setText("About to load URL");
}
protected void onProgressUpdate(
String... values) {
// just one value, please
mStatus.setText(values[0]);
}}
```

When launched with the *AsyncTask.execute()* method, *doInBackground()* runs in a background thread while the other methods run on the UI thread. There is no need to manage a Handler or post a Runnable object to it. This simplifies coding and debugging.

## 12.8 Using Threads for Network Calls :

The following code demonstrates how to launch a new thread that connects to a remote server, retrieves and parses some XML, and posts a response back to the UI thread to change a *TextView* :

```
import java.net.URL;
import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserFactory;
// ...
new Thread() {
public void run() {
try {
URL text = new URL(
"http://api.flickr.com/services/feeds/
photos_public.gne?
    ➥id=26648248@N04&lang=en-us&format=atom");
XmlPullParserFactory parserCreator =
XmlPullParserFactory.newInstance();
XmlPullParser parser =
parserCreator.newPullParser();
parser.setInput(text.openStream(), null);
mHandler.post(new Runnable() {
public void run() {
status.setText("Parsing...");
}
});
int parserEvent = parser.getEventType();
while (parserEvent !=
XmlPullParser.END_DOCUMENT) {
// Parsing code here ...
parserEvent = parser.next();
}
mHandler.post(new Runnable() {
public void run() {
status.setText("Done...");
}
});
} catch (Exception e) {
Log.e("Net", "Error in network call", e);
}
}
}.start();
```

For this example, an anonymous Thread object will do. We create it and call its *start()* method immediately. However, now that the code runs on a separate thread, the user interface updates must be posted back to the main thread. This is done by using a Handler object on the main thread and creating Runnable objects that execute to call *setText()* on the *TextView* widget named status.

The rest of the code remains the same as in the previous examples. Executing both the parsing code and the networking code on a separate thread allows the user interface to continue to behave in a responsive fashion while the network and parsing operations are done behind the scenes, resulting in a smooth and friendly user experience. This also all ows for handling of interim actions by the user, such as canceling the transfer. You can accomplish this by implementing the Thread to listen for certain events and check for certain flags.

## 12.9 Displaying Images from a Network Resource :

Now that we have covered how you can use a separate thread to parse XML, let's take our example a bit deeper and talk about working with non–primitive data types. Continuing with the previous example of parsing for image locations from a flickr feed, let's display some images from the feed. The following example reads the image data and displays it on the screen, demonstrating another way you can use network resources :

```
import java.io.InputStream;
import java.net.URL;
import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserFactory;
import android.os.Handler;
// ...
final String imageSrc =
parser.getAttributeValue(null, "href");
final String currentTitle = new String(title);
imageThread.queueEvent(new Runnable() {
```

```
public void run() {
InputStream bmis;
try {
bmis = new URL(imageSrc).openStream();
final Drawable image = new BitmapDrawable(
BitmapFactory.decodeStream(bmis));
mHandler.post(new Runnable() {
public void run() {
imageSwitcher.setImageDrawable(image);
info.setText(currentTitle);
}
});
} catch (Exception e) {
Log.e("Net", "Failed to grab image", e);
}
}
});
```

You can find this block of code within the parser thread, as previously described. After the image source and title of the image have been determined, a new Runnable object is queued for execution on a separate image handling thread. The thread is merely a queue that receives the anonymous Runnable object created here and executes it at least 10 seconds after the last one, resulting in a slideshow of the images from the feed.

As with the first networking example, a new URL object is created and an InputStream retrieved from it. You need a Drawable object to assign to the ImageSwitcher.Then you use the BitmapFactory.decodeStream() method, which takes an InputStream. Finally, from this Runnable object, which runs on a separate queuing thread, spacing out image drawing, another anonymous Runnable object posts back to the main thread to actually update the ImageSwitcher with the new image. Figure 12.1 shows what the screen might look like showing decoding status and displaying the current image.

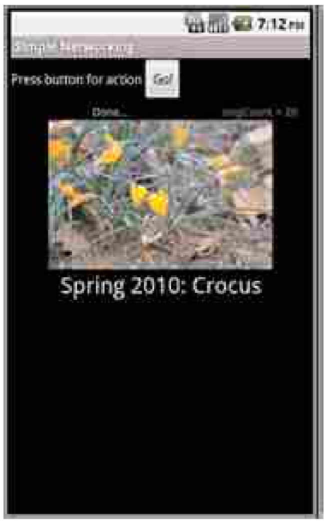


Figure 12.1 : Screen showing a flickr image and decoding status of feed.

Although all this continues to happen while the feed from flickr is decoded, certain operations are slower than others. For instance, while the image is decoded or drawn on the screen, you can notice a distinct hesitation in the progress of the decoding. This is to be expected on current mobile devices because most have only a single thread of execution available for applications. You need to use careful design to provide a reasonably smooth and responsive experience to the user.

## 12.10 Retrieving Android Network Status :

The Android SDK provides utilities for gathering information about the current state of the network. This is useful to determine if a network connection is even available before trying to use a network resource. The *ConnectivityManager* class provides a number of methods to do this. The following code determines if the mobile (cellular) network is available and connected. In addition, it determines the same for the Wi–Fi network:

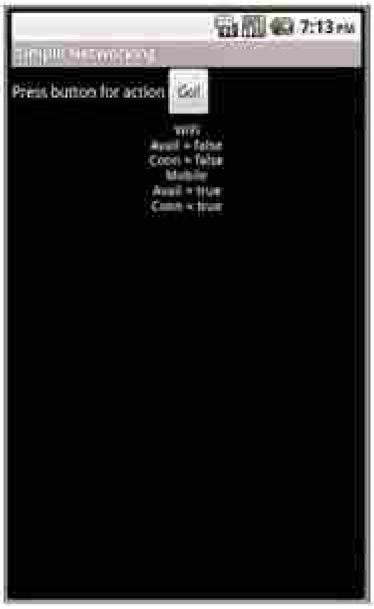
```
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
// ...
ConnectivityManager cm = (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo ni =
cm.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
boolean isWifiAvail = ni.isAvailable();
boolean isWifiConn = ni.isConnected();
ni = cm.getNetworkInfo(ConnectivityManager.TYPE_
MOBILE);
boolean isMobileAvail = ni.isAvailable();
boolean isMobileConn = ni.isConnected();
status.setText("WiFi\nAvail = "+ isWifiAvail +
"\nConn = " + isWifiConn +
"\nMobile\nAvail = "+ isMobileAvail +
"\nConn = " + isMobileConn);
```

First, an instance of the *ConnectivityManager* object is retrieved with a call to the *getSystemService()* method, available as part of your application Context. Then this instance retrieves *NetworkInfo* objects for both TYPE_WIFI and TYPE_MOBILE (for the cellular network). These objects are queried for their availability but can also be queried at a more detailed status level to learn exactly what state of connection (or disconnection) the network is in. Figure 12.2 shows the typical output for the emulator in which the mobile network is simulated but Wi–Fi isn't available.

If the network is available, this does not necessarily mean the server that the network resource is on is available. However, a call to the *ConnectivityManager* method *requestRouteToHost()* can answer this question. This way, the application can give the user better feedback when there are network problems.

For your application to read the status of the network, it needs explicit permission. The following statement is required to be in its AndroidManifest.xml file :

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



```
7:13 PM
Press button for action  Go!
WiFi
Avail = false
Conn = false
Mobile
Avail = true
Conn = true
```

**Figure 12.2 : Typical network status of the Android SDK emulator.**

## ❑ Check Your Progress :

1. Which package should be imported to use Android Networking API ?

   (A) import java.awt.*; (B) import java.util.*;

   (C) import java.io.*; (D) import java.net.*;

2. HTTP stands for ________

   (A) Hypertexture Transfer Protocol

   (B) Hypertext Transfer Protocol

   (C) Hypertext Transportation Protocol

   (D) Hypertension Transfer Protocol

3. ________ retrieves some information about the resource referenced by the URL object, including HTTP status and header information

   (A) HttpURLConnection (B) WebHttpURLConnection

   (C) XmlURLConnection (D) WebURLConnection

4. Which of the following class in android executes the task asynchronously with your service ?

   (A) SyncTask (B) AsyncTask

   (C) WebTask (D) AsyncronousTask

5. ________ Parser to read an XML file

   (A) HTML Pull (B) Web Pull

   (C) XML Pull (D) Text Pull

## 12.11 Let Us Sum Up :

In this unit we discussed regarding to learn how to use Network API in Android Application, to learn how to Access the Internet (HTTP), to understand how to Read Data from the Web, to learn parsing the XML Data and to understand how to get Network status

## 12.12 Answers for Check Your Progress :

**1.** (D) **2.** (B) **3.** (A) **4.** (B) **5.** (C)

## 12.13 Glossary :

**1.** **SDK :** Software Development Kit

**2.** **AVD :** Android Virtual Device

**3.** **DDMS :** Dalvik Debug Monitor Server

**4.** **ADB :** Android Debug Bridge

## 12.14 Assignment :

1. What is Android networking ?
2. What are network libraries in Android ?
3. What is networking on phone ?
4. Explain in details Android Networking API with its supported methods.

## 12.15 Activities :

Search the nearby firm who require android application with network facility and analysis it and collect data for further development of the application.

## 12.16 Case Study :

Create Network enabled Android Application with internet access.

## 12.17 Further Reading :

Lauren Darcey and Shane Conder, "Android Wireless Application Development", Pearson Education, 2nd ed. (2011).

Unit 13

# USING ANDROID WEB APIS & TELEPHONY APIS

## UNIT STRUCTURE

## 13.0 Learning Objectives :

- To learn how to use Web API in Android Application
- To learn how to Design Layout with a WebView Control
- To understand how to apply WebKit API
- To learn and create Flash Application Using Android
- To lean about Telephony API in Android Application
- To understand and create mobile application with calling a number
- To understand and create mobile application to send the message on number

## 13.1 Introduction :

Mobile developers often rely upon web technologies to enrich their applications, provide fresh content, and integrate with popular web services such as social networks. Android application can harness the power of the Internet in a variety of ways, including adding browser functionality to applications using the special WebView control and extending web–based functionality using standard WebKit libraries. Newer Android devices can also run Flash applications. In this unit, we discuss the web technologies available on the Android platform.

## 13.2 Browsing the Web with WebView :

Applications that retrieve and display content from the Web often end up displaying that data on the screen. Instead of customizing various screens with custom controls, Android applications can simply use the WebView control to display web content to the screen. You can think of the WebView control as a browser–like view. The WebView control uses the WebKit rendering engine to draw HTML content on the screen. This content could be HTML pages on the Web or it can be locally sourced. WebKit is an open source browser engine. You can read more about it on its official website at http://webkit.org.

Using the WebView control requires the android.permission.INTERNET permission. You can add this permission to your application's Android manifest file as follows :

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

When deciding if the WebView control is right for your application, consider that you can always launch the Browser application using an Intent. When you want the user to have full access to all Browser features, such as bookmarking and browsing, you're better off launching into the Browser application to a specific website, letting users do their browsing, and having them return to your application when they're done. You can do this as follows :

```
Uri uriUrl = Uri.parse("http://androidbook.
blogspot.com/");
Intent launchBrowser = new Intent(Intent.ACTION_VIEW,
uriUrl);
startActivity(launchBrowser);
```

Launching the Browser via an Intent does not require any special permissions. This means that your application is not required to have the android.permission.INTERNET permission.

In addition, because Android transitions from your application's current activity to a specific Browser application's activity, and then returns when the user presses the back key, the experience is nearly as seamless as implementing your own Activity class with an embedded WebView.

### 13.2.1 Designing a Layout with a WebView Control :

The WebView control can be added to a layout resource file like any other view. It can take up the entire screen or just a portion of it. A typical WebView definition in a layout resource might look like this :

```
<WebView
android:id="@+id/web_holder"
android:layout_height="wrap_content"
android:layout_width="fill_parent"
/>
```

Generally speaking, you should give your WebView controls ample room to display text and graphics. Keep this in mind when designing layouts using the WebView control.

### 13.2.2 Loading Content into a WebView Control :

You can load content into a WebView control in a variety of ways. For example, a WebView control can load a specific website or render raw HTML content. Web pages can be stored on a remote web server or stored on the device.

Here is an example of how to use a WebView control to load content from a specific website :

```
final WebView wv = (WebView) findViewById(R.id.web_
holder);
wv.loadUrl("http://www.perlgurl.org/");
```

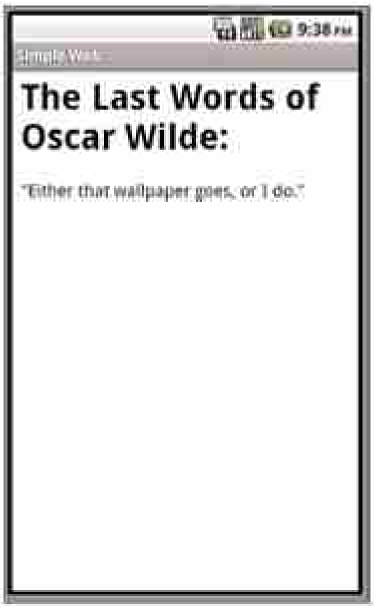
You do not need to add any additional code to load the referenced web page on the screen. Similarly, you could load an HTML file called webby.html stored in the application's assets directory like this :

```
wv.loadUrl("file:///android_asset/webby.html");
```

If, instead, you want to render raw HTML, you can use the loadData() method :

```
String strPageTitle = "The Last Words of Oscar Wilde";
String strPageContent = "<h1>" + strPageTitle +
": </h1>\"Either that wallpaper goes, or I do.\"";
String myHTML = "<html><title>" + strPageTitle
+"</title><body>"+ strPageContent +"</body></html>";
wv.loadData(myHTML, "text/html", "utf-8");
```

The resulting WebView control is shown in Figure 13.1.



The Last Words of Oscar Wilde:

"Either that wallpaper goes, or I do."

**Figure 13.1 : A WebView control used to display HTML.**

Unfortunately, not all websites are designed for mobile devices. It can be handy to change the scale of the web content to fit comfortably within the WebView control. You can achieve this by setting the initial scale of the control, like this :

```
wv.setInitialScale(30);
```

The call to the *setInitialScale()* method scales the view to 30 percent of the original size. For pages that specify absolute sizes, scaling the view is necessary to see the entire page on the screen. Some text might become too small to read, though, so you might need to test and make page design changes (if the web content is under your control) for a good user experience.

### 13.2.3 Adding Features to the WebView Control :

You might have noticed that the WebView control does not have all the features of a full browser. For example, it does not display the title of a webpage or provide buttons for reloading pages. In fact, if the user clicks on a link within the *WebView* control, that action does not load the new page within the view. Instead, it fires up the Browser application. By default, all the *WebView* control does is display the web content provided by the developer using its internal rendering engine, *WebKit*. You can enhance the WebView control in a variety of ways, though. You can use three classes, in particular, to help modify the behavior of the control: the WebSettings class, the *WebViewClient* class, and the *WebChromeClient* class.

#### Modifying WebView Settings with WebSettings

By default, a *WebView* control has various default settings: no zoom controls, JavaScript disabled, default font sizes, user–agent string, and so on. You can change the settings of a *WebView* control using the *getSettings()* method. The *getSettings()* method returns a *WebSettings* object that can be used to configure the desired *WebView* settings. Some useful settings include

- Enabling and disabling zoom controls using the setSupportZoom() and setBuiltInZoomControls() methods
- Enabling and disabling JavaScript using the setJavaScriptEnabled() method
- Enabling and disabling mouseovers using the setLightTouchEnabled() method
- Configuring font families, text sizes, and other display characteristics

You can also use the WebSettings class to configure WebView plug–ins and allow for multiple windows.

#### Handling WebView Events with WebViewClient

The *WebViewClient* class enables the application to listen for certain WebView events, such as when a page is loading, when a form is submitted, and when a new URL is about to be loaded. You can also use the WebViewClient class to determine and handle any errors that occur with page loading. You can tie a valid WebViewClient object to a WebView using the setWebViewClient() method.

The following is an example of how to use *WebViewClient* to handle the *onPageFinished()* method to draw the title of the page on the screen :

```
WebViewClient webClient = new WebViewClient() {
    public void onPageFinished(WebView view, String url) {
super.onPageFinished(view, url);
String title = wv.getTitle();
pageTitle.setText(title);
}};
wv.setWebViewClient(webClient);
```

When the page finishes loading, as indicated by the call to *onPageFinished()*, a call to the *getTitle()* method of the WebView object retrieves the title for use. The result of this call is shown in Figure 13.2.

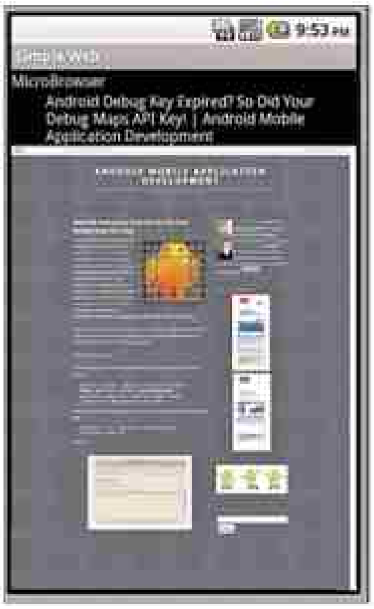

9:53 PM
MicroBrowser
Android Debug Key Expired? So Did Your Debug Maps API Key! | Android Mobile Application Development

**Figure 13.2 : A WebView control with microbrowser features such as title display**

**Adding Browser Chrome with WebChromeClient :**

You can use the *WebChromeClient* class in a similar way to the *WebViewClient*. However, *WebChromeClient* is specialized for the sorts of items that will be drawn outside the region in which the web content is drawn, typically known as browser chrome. The *WebChromeClient* class also includes callbacks for certain JavaScript calls, such as *onJsBeforeUnload()*, to confirm navigation away from a page. A valid *WebChromeClient* object can be tied to a WebView using the *setWebChromeClient()* method.

The following code demonstrates using WebView features to enable interactivity with the user. An EditText and a Button control are added below the WebView control, and a Button handler is implemented as follows :

```
Button go = (Button) findViewById(R.id.go_button);
go.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
wv.loadUrl(et.getText().toString());
}
});
```

Calling the *loadUrl()* method again, as shown, is all that is needed to cause the WebView control to download another HTML page for display, as shown in Figure 13.3. From here, you can build a generic web browser in to any application, but you can apply restrictions so that the user is restricted to browsing relevant materials.

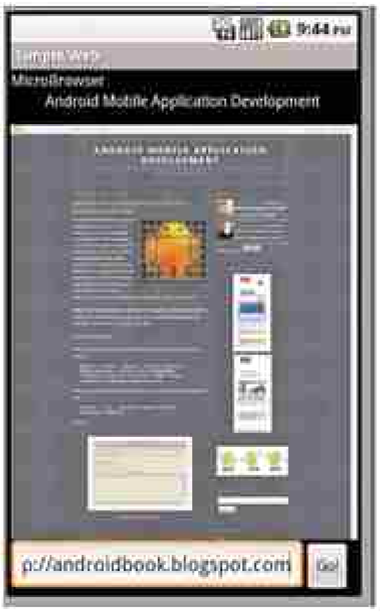

MicroBrowser
Android Mobile Application Development
p://androidbook.blogspot.com
Go

**Figure 13.3 : WebView with EditText allowing entry of arbitrary URLs.**

Using WebChromeClient can help add some typical chrome on the screen. For instance, you can use it to listen for changes to the title of the page, various JavaScript dialogs that might be requested, and even for developer–oriented pieces, such as the console messages.

```
WebChromeClient webChrome = new WebChromeClient() {
@Override
public void onReceivedTitle
(WebView view, String title) {
Log.v(DEBUG_TAG, "Got new title");
super.onReceivedTitle(view, title);
pageTitle.setText(title);
}
};
wv.setWebChromeClient(webChrome);
```

Here the default *WebChromeClient* is overridden to receive changes to the title of the page. This title of the web page is then set to a *TextView* visible on the screen. Whether you use WebView to display the main user interface of your application or use it sparingly to draw such things as help pages, there are circumstances where it might be the ideal control for the job to save coding time, especially when compared to a custom screen design. Leveraging the power of the open source engine, WebKit, WebView can provide a powerful, standards–based HTML viewer for applications. Support for WebKit is widespread because it is used in various desktop browsers, including Apple Safari and Google Chrome, a variety of mobile browsers, including those on the Apple iOS, Nokia, Palm WebOS, and BlackBerry handsets, and various other platforms, such as Adobe AIR.

## 13.3 Building Web Extensions Using WebKit :

All HTML rendering on the Android platform is done using the *WebKit* rendering engine. The android.webkit package provides a number of APIs for browsing the Internet using the powerful WebView control. You should be aware of the *WebKit* interfaces and classes available, as you are likely to need them to enhance the WebView user experience. These are not classes and interfaces to the Browser app (although you can interact with the Browser data using contact providers). Instead, these are the classes and interfaces that you must use to control the browsing abilities of WebView controls you implement in your applications.

### 13.3.1 Browsing the WebKit APIs :

Some of the most helpful classes of the android.webkit package are

- The *CacheManager* class gives you some control over cache items of a WebView.
- The *ConsoleMessage* class can be used to retrieve JavaScript console output from a WebView.
- The *CookieManager* class is used to set and retrieve user cookies for a WebView.
- The *URLUtil* class is handy for validating web addresses of different types.
- The WebBackForwardList and WebHistoryItem classes can be used to inspect the web history of the WebView.

Now let's take a quick look at how you might use some of these classes to enhance a *WebView*.

### 13.3.2 Extending Web Application Functionality to Android :

Let's take some of the WebKit features we have discussed so far in this chapter and work through an example. It is fairly common for mobile developers to design their applications as web applications in order to reach users across a variety of platforms. This minimizes the amount of platform–specific code to develop and maintain. However, on its own, a web application cannot call into native platform code and take advantage of the features that native apps (such as those written in Java for the Android platform) can, such as using a built–in camera or accessing some other underlying Android feature. Developers can enhance web applications by designing a lightweight shell application in Java and using a WebView control as a portal to the web application content. Two–way communication between the web application and the native Java application is possible through scripting languages such as JavaScript.

Let's create a simple Android application that illustrates communication between web content and native Android code. This example requires that you understand JavaScript.

To create this application, take the following steps :

1. Create a new Android application.
2. Create a layout with a WebView control called html_viewer and a Button control called *call*_js. Set the *onClick* attribute of the Button control to a method called setHTMLText.
3. In the *onCreate()* method of your application activity, retrieve the *WebView* control using the *findViewById()* method.
4. Enable JavaScript within the WebView by retrieving its *WebSettings* and calling the *setJavaScriptEnabled()* method.

5. Create a *WebChromeClient* object and implement its *onConsoleMessage()* method in order to monitor the JavaScript console messages.
6. Add the WebChromeClient object to the WebView using the *setWebChromeClient()* method.
7. Allow the JavaScript interface to control your application by calling the *addJavascriptInterface()* method of the WebView control. You will need to define the functionality that you want the JavaScript interface to be able to control and within what namespace the calls will be available. In this case, we allow the JavaScript to initiate Toast messages.
8. Load your content into the WebView control using one of the standard methods, such as the *loadUrl()* method. In this case, we load an HTML asset we defined within the application package.

If you followed these steps, you should end up with your activity's onCreate() method looking something like this :

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
final WebView wv = (WebView) findViewById(R.id.html_
    viewer);
WebSettings settings = wv.getSettings();
settings.setJavaScriptEnabled(true);
WebChromeClient webChrome = new WebChromeClient() {
@Override
public boolean onConsoleMessage(ConsoleMessage
    consoleMessage) {
Log.v(DEBUG_TAG, consoleMessage.lineNumber()
+ ": " + consoleMessage.message());
return true;
}
};
wv.setWebChromeClient(webChrome);
wv.addJavascriptInterface(new JavaScriptExtensions(),
    "jse");
wv.loadUrl("file:///android_asset/sample.html");
}
```

A custom *WebChromeClient* class is set so that any JavaScript console.log messages go out to *LogCat* output, using a custom debug tag as usual to enable easy tracking of log output specific to the application. Next, a new JavaScript interface is defined with the namespace called jse–the namespace is up to you. To call from JavaScript to this Java class, the JavaScript calls must all start with namespace jse., followed by the appropriate exposed method–for instance, jse.javaMethod().

You can define the *JavaScriptExtensions* class as a subclass within the activity as a subclass with a single method that can trigger Android Toast messages :

```
class JavaScriptExtensions {
public static final int TOAST_LONG = Toast.LENGTH_LONG;
public static final int TOAST_SHORT = Toast.LENGTH_
    SHORT;
public void toast(String message, int length) {
Toast.makeText(SimpleWebExtension.this, message,
    length).show();
}
}
```

The JavaScript code has access to everything in the JavaScriptExtensions class, including the member variables as well as the methods. Return values work as expected from the methods, too.

Now switch your attention to defining the web page to load in the WebView control.

For this example, simply create a file called sample.html in the /assets directory of the application.

The contents of the sample.html file are shown here :

```
<html>
<head>
<script type="text/javascript">
function doToast() {
jse.toast("'"+document.getElementById('form_
text').value +
"' -From Java!", jse.TOAST_LONG);
}
function doConsoleLog() {
console.log("Console logging.");
}
function doAlert() {
alert("This is an alert.");
}
function doSetFormText(update) {
document.getElementById('form_text').value = update;
}
</script>
</head>
<body>
<h2>This is a test.</h2>
```

```
<input type="text" id="form_text" value="Enter
    something here..." />
<input type="button" value="Toast" onclick="doToast();
    " /><br />
<input type="button" value="Log" onclick=
    "doConsoleLog();" /><br />
<input type="button" value="Alert" onclick=
    "doAlert();" />
</body>
</html>
```

The sample.html file defines four JavaScript functions and displays the form shown within the WebView :

- The *doToast()* function calls into the Android application using the jse object defined earlier with the call to the addJavaScriptInterface() method. The addJavaScriptInterface() method, for all practical intents and purposes, can be treated literally as the JavaScriptExtensions class as if that class had been written in JavaScript. If the doToast() function had returned a value, we could assign it to a variable here.
- The *doConsoleLog()* function writes into the JavaScript console log, which is picked up by the onConsoleMessage() callback of the WebChromeClient.
- The *doAlert()* function illustrates how alerts work within the WebView control by launching a dialog. If you want to override what the alert looks like, you can override the WebChromeClient.onJSAlert() method.
- The *doSetFormText()* function illustrates how native Java code can communicate back through the JavaScript interface and provide data to the web application. Finally, to demonstrate making a call from Java back to JavaScript, you need to define the click handler for the Button control within your Activity class. Here, the onClick handler, called setHTMLText(), executes some JavaScript on the currently loaded page by calling a JavaScript function called doSetFormText(), which we defined earlier in the web page.

  Here is an implementation of the setHTMLText() method :

```
public void setHTMLText(View view) {
WebView wv = (WebView) findViewById(R.id.html_viewer);
wv.loadUrl("javascript:doSetFormText('Java->JS
    call');");
}
```

This method of making a call to the JavaScript on the currently loaded page does not allow for return values. There are ways, however, to structure your design to allow checking of results, generally by treating the call as asynchronous and implementing another method for determining the response.

## 13.4 Using Android Telephony APIs :

Although the Android platform has been designed to run on almost any type of device, the Android devices available on the market are primarily phones. Applications can take advantage of this fact by integrating phone features into their feature set. This chapter introduces you to the telephony–related APIs available within the Android SDK.

### 13.4.1 Working with Telephony Utilities :

The Android SDK provides a number of useful utilities for applications to integrate phone features available on the device. Generally speaking, developers should consider an Android device first and foremost as a phone. Although these devices might also run applications, phone operations generally take precedence. Your application should not interrupt a phone conversation, for example. To avoid this kind of behavior, your application should know something about what the user is doing, so that it can react differently. For instance, an application might query the state of the phone and determine that the user is talking on the phone and then choose to vibrate instead of play an alarm. In other cases, applications might need to place a call or send a text message. Phones typically support a Short Message Service (SMS), which is popular for texting (text messaging). Enabling the capability to leverage this feature from an application can enhance the appeal of the application and add features that can't be easily replicated on a desktop environment. Because many Android devices are phones, applications frequently deal with phone numbers and the contacts database; some might want to access the phone dialer to place calls or check phone status information. Adding telephony features to an application enables a more integrated user experience and enhances the overall value of the application to the users.

**Gaining Permission to Access Phone State Information**

Let's begin by looking at how to determine telephony state of the device, including the ability to request the hook state of the phone, information of the phone service, and utilities for handling and verifying phone numbers. The *TelephonyManager* object within the *android.telephony* package is a great place to start. Many of the method calls in this section require explicit permission set with the Android application manifest file. The READ_PHONE_STATE permission is required to retrieve information such as the call state, handset phone number, and device identifiers or serial numbers. The ACCESS_COARSE_LOCATION permission is required for cellular location information.

The following block of XML is typically needed in your application's AndroidManifest.xml file to access basic phone state information :

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

**Requesting Call State**

You can use the TelephonyManager object to retrieve the state of the phone and some information about the phone service itself, such as the phone number of the handset.

You can request an instance of *TelephonyManager* using the getSystemService() method, like this :

```
TelephonyManager telManager = (TelephonyManager)
getSystemService(Context.TELEPHONY_SERVICE);
```

With a valid TelephonyManager instance, an application can now make several queries.

One important method is getCallState().This method can determine the voice call status of the handset. The following block of code shows how to query for the call state and all the possible return values:

```
int callStatus = telManager.getCallState();
String callState = null;
switch (callStatus) {
case TelephonyManager.CALL_STATE_IDLE:
callState = "Phone is idle.";
break;
case TelephonyManager.CALL_STATE_OFFHOOK:
callState = "Phone is in use.";
break;
case TelephonyManager.CALL_STATE_RINGING:
callState = "Phone is ringing!";
break;
}
Log.i("telephony", callState);
```

The three call states can be simulated with the emulator through the Dalvik Debug Monitor Service (DDMS) tool, which is discussed in detail in Appendix B, "The Android DDMS Quick–Start Guide."

Querying for the call state can be useful in certain circumstances. However, listening for changes in the call state can enable an application to react appropriately to something the user might be doing. For instance, a game might automatically pause and save state information when the phone rings so that the user can safely answer the call. An application can register to listen for changes in the call state by making a call to the listen() method of TelephonyManager.

```
telManager.listen(new PhoneStateListener() {
public void onCallStateChanged(
int state, String incomingNumber) {
String newState = getCallStateString(state);
if (state == TelephonyManager.CALL_STATE_RINGING) {
Log.i("telephony", newState +
" number = " + incomingNumber);
} else {
Log.i("telephony", newState);
}
}
}, PhoneStateListener.LISTEN_CALL_STATE);
```

The listener is called, in this case, whenever the phone starts ringing, the user makes a call, the user answers a call, or a call is disconnected. The listener is also called right after it is assigned so an application can get the initial state.

Another useful state of the phone is determining the state of the service. This information can tell an application if the phone has coverage at all, if it can only make emergency calls, or if the radio for phone calls is turned off as it might be when in airplane mode. To do this, an application can add the PhoneStateListener.LISTEN_SERVICE_STATE flag to the listener described earlier and implement the onServiceStateChanged method, which receives an instance of the ServiceState object. Alternatively, an application can check the state by constructing a ServiceState object and querying it directly, as shown here :

```
int serviceStatus = serviceState.getState();
String serviceStateString = null;
switch (serviceStatus) {
case ServiceState.STATE_EMERGENCY_ONLY:
serviceStateString = "Emergency calls only";
break;
case ServiceState.STATE_IN_SERVICE:
serviceStateString = "Normal service";
break;
case ServiceState.STATE_OUT_OF_SERVICE:
serviceStateString = "No service available";
break;
case ServiceState.STATE_POWER_OFF:
serviceStateString = "Telephony radio is off";
break;
}
Log.i("telephony", serviceStateString);
```

In addition, a status such as whether the handset is roaming can be determined by a call to the getRoaming() method. A friendly and frugal application can use this method to warn the user before performing any costly roaming operations such as data transfers within the application.

**Requesting Service Information**

In addition to call and service state information, your application can retrieve other information about the device. This information is less useful for the typical application but can diagnose problems or provide specialized services available only from certain provider networks.

The following code retrieves several pieces of service information :

```
String opName = telManager.getNetworkOperatorName();
Log.i("telephony", "operator name = " + opName);
String phoneNumber = telManager.getLine1Number();
```

```
Log.i("telephony", "phone number = " + phoneNumber);
String providerName = telManager.getSimOperatorName();
Log.i("telephony", "provider name = " + providerName);
```

The network operator name is the descriptive name of the current provider that the handset connects to. This is typically the current tower operator. The SIM operator name is typically the name of the provider that the user is subscribed to for service. The phone number for this application programming interface (API) is defined as the MSISDN, typically the directory number of a GSM handset (that is, the number someone would dial to reach that particular phone).

**Monitoring Signal Strength and Data Connection Speed**

Sometimes an application might want to alter its behavior based upon the signal strength or service type of the device. For example, a high–bandwidth application might alter stream quality or buffer size based on whether the device has a low–speed connection (such as 1xRTT or EDGE) or a high–speed connection (such as EVDO or HSDPA).

TelephonyManager can be used to determine such information.

If your application needs to react to changes in telephony state, you can use the listen() method of TelephonyManager and implement a PhoneStateListener to receive changes in service, data connectivity, call state, signal strength, and other phone state information.

**Working with Phone Numbers**

Applications that deal with telephony, or even just contacts, frequently have to deal with the input, verification, and usage of phone numbers. The Android SDK includes a set of helpful utility functions that simplify handling of phone number strings. Applications can have phone numbers formatted based on the current locale setting. For example, the following code uses the formatNumber() method :

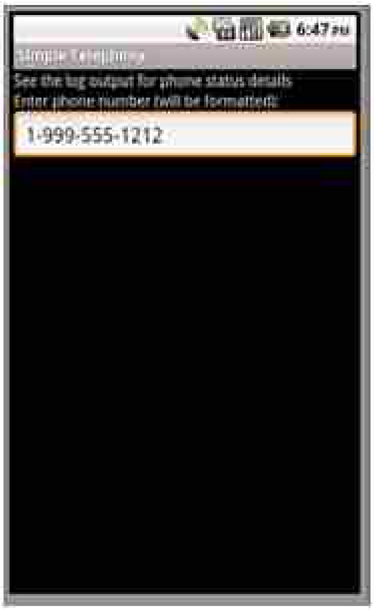
```
String formattedNumber =
PhoneNumberUtils.formatNumber("9995551212");
Log.i("telephony", formattedNumber);
```

The resulting output to the log would be the string 999–555–1212 in my locale. Phone numbers can also be compared using a call to the *PhoneNumberUtils.compare()* method. An application can also check to see if a given phone number is an emergency phone number by calling *PhoneNumberUtils.isEmergencyNumber()*, which enables your application to warn users before they call an emergency number. This method is useful when the source of the phone number data might be questionable.

The *formatNumber()* method can also take an Editable as a parameter to format a number in place. The useful feature here is that you can assign the PhoneNumberFormattingTextWatcher object to watch a TextView (or EditText for user input) and format phone numbers as they are entered. The following code demonstrates the ease of configuring an EditText to format phone numbers that are entered :

```
EditText numberEntry = (EditText)
findViewById(R.id.number_entry);
numberEntry.addTextChangedListener(
new PhoneNumberFormattingTextWatcher());
```

While the user is typing in a valid phone number, the number is formatted in a way suitable for the current locale. Just the numbers for 19995551212 were entered on the EditText shown in Figure 13.4.



See the log output for phone status details
Enter phone number (will be formatted):
1-999-555-1212

**Figure 13.4 : Screen showing formatting results after entering only digits.**

## 13.5 Using SMS :

SMS usage has become ubiquitous in the last several years. Integrating messaging services, even if only outbound, to an application can provide familiar social functionality to the user. SMS functionality is provided to applications through the *android.telephony* package.

### Gaining Permission to Send and Receive SMS Messages

SMS functionality requires two different permissions, depending on if the application sends or receives messages. The following XML, to be placed with AndroidManifest.xml, shows the permissions needed for both actions :

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

### Sending an SMS

To send an SMS, an application first needs to get an instance of the SmsManager. Unlike other system services, this is achieved by calling the static method getDefault() of SmsManager :

```
final SmsManager sms = SmsManager.getDefault();
```

Now that the application has the SmsManager, sending SMS is as simple as a single call :

```
sms.sendTextMessage(
"9995551212", null, "Hello!", null, null);
```

The application does not know if the actual sending of the SMS was successful without providing a PendingIntent to receive the broadcast of this information. The following code demonstrates configuring a PendingIntent to listen for the status of the SMS :

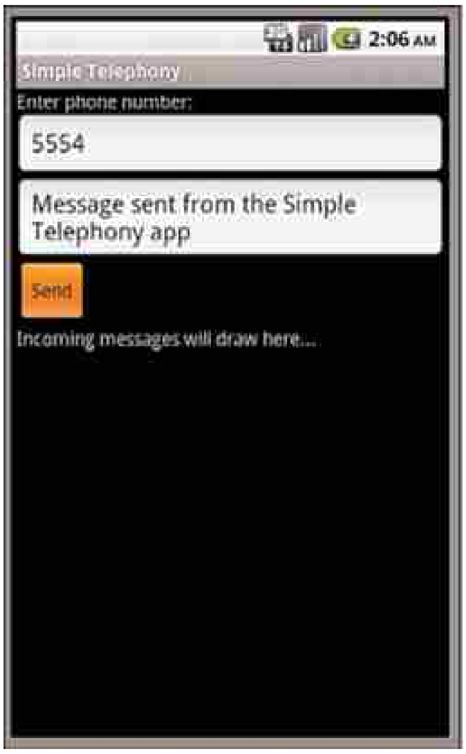
```
Intent msgSent = new Intent("ACTION_MSG_SENT");
final PendingIntent pendingMsgSent =
PendingIntent.getBroadcast(this, 0, msgSent, 0);
registerReceiver(new BroadcastReceiver() {
public void onReceive(Context context, Intent intent) {
int result = getResultCode();
if (result != Activity.RESULT_OK) {
Log.e("telephony",
"SMS send failed code = " + result);
pendingMsgReceipt.cancel();
} else {
messageEntry.setText("");
}
}
}, new IntentFilter("ACTION_MSG_SENT"));
```

The PendingIntent pendingMsgSent can be used with the call to the sendTextMessage(). The code for the message–received receipt is similar but is called when the sending handset receives acknowledgment from the network that the destination handset received the message.

If we put all this together with the preceding phone number formatting EditText, a new entry field for the message, and a button,we can create a simple form for sending an SMS message. The code for the button handling looks like the following :

```
Button sendSMS = (Button) findViewById(R.id.send_sms);
sendSMS.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
String destination =
numberEntry.getText().toString();
String message =
messageEntry.getText().toString();
sms.sendTextMessage(destination, null, message,
pendingMsgSent, pendingMsgReceipt);
registerReceiver(...);
}
}
```

After this code is hooked in, the result should look something like Figure 13.5. Within this application, we used the emulator "phone number" trick (its port number). This is a great way to test sending SMS messages without using hardware or without incurring charges by the handset operator.



2:06 AM
Simple Telephony
Enter phone number:
5554
Message sent from the Simple Telephony app
Send
Incoming messages will draw here...

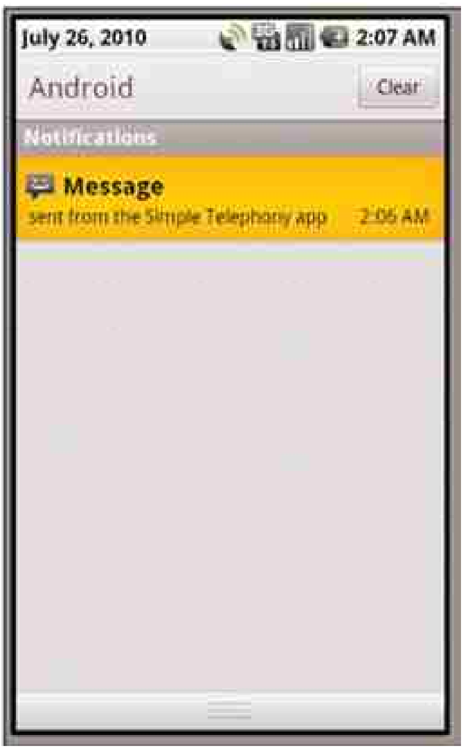


**Figure 13.5 : Two emulators, one sending an SMS from an application and one receiving an SMS.**

A great way to extend this would be to set the sent receiver to modify a graphic on the screen until the sent notification is received. Further, you could use another graphic to indicate when the recipient has received the message. Alternatively, you could use *ProgressBar* widgets track the progress to the user.

**Receiving an SMS**

Applications can also receive SMS messages. To do so, your application must register a *BroadcastReceiver* to listen for the Intent action associated with receiving an SMS. An application listening to SMS in this way doesn't prevent the message from getting to other applications.

Expanding on the previous example, the following code shows how any incoming text message can be placed within a *TextView* on the screen :

```
final TextView receivedMessage =
(TextView)findViewById(
R.id.received_message);
rcvIncoming = new BroadcastReceiver() {
public void onReceive(Context context, Intent intent) {
Log.i("telephony", "SMS received");
Bundle data = intent.getExtras();
if (data != null) {
Object pdus[] =
(Object[]) data.get("pdus");
String message = "New message:\n";
String sender = null;
for (Object pdu : pdus) {
SmsMessage part = SmsMessage.
createFromPdu((byte[])pdu);
message += part.
getDisplayMessageBody();
if (sender == null) {
sender = part.
getDisplayOriginatingAddress();
}
}
receivedMessage.setText(
message + "\nFrom: "+sender);
numberEntry.setText(sender);
}
}
};
registerReceiver(rcvIncoming, new IntentFilter(
"android.provider.Telephony.SMS_RECEIVED"));
```

This block of code is placed within the onCreate() method of the Activity. First, the message Bundle is retrieved. In it, an array of Objects holds several byte arrays that contain PDU data–the data format that is customarily used by wireless messaging protocols.

Luckily, the Android SDK can decode these with a call to the static *SmsMessage*.createFromPdu() utility method. From here, we can retrieve the body of the SMS message by calling *getDisplayMessageBody()*.

The message that comes in might be longer than the limitations for an SMS. If it is, it will have been broken up in to a multipart message on the sending side. To handle this,we loop through each of the received Object parts and take the corresponding body from each, while only taking the sender address from the first.

Next, the code updates the text string in the *TextView* to show the user the received message. The sender address is also updated so that the recipient can respond with less typing.

Finally, we register the *BroadcastReceiver* with the system. The *IntentFilter* used here, *android.provider.Telephony.SMS_RECEIVED*, is a well–known but undocumented *IntentFilter* used for this. As such, we have to use the string literal for it.

## 13.6 Making and Receiving Phone Calls :

It might come as a surprise to the younger generation (they usually just text), but phones are often still used for making and receiving phone calls. Any application can be made to initiate calls and answer incoming calls; however, these abilities should be used judiciously so as not to unnecessarily disrupt the calling functionality of the user's device.

**Making Phone Calls**

You've seen how to find out if the handset is ringing. Now let's look at how to enable your application to make phone calls as well.

Building on the previous example, which sent and received SMS messages, we now walk through similar functionality that adds a call button to the screen to call the phone number instead of messaging it.

The Android SDK enables phone numbers to be passed to the dialer in two different ways. The first way is to launch the dialer with a phone number already entered. The user then needs to press the Send button to actually initiate the call. This method does not require any specific permissions. The second way is to actually place the call. This method requires the *android.permission.CALL_PHONE* permission to be added to the application's *AndroidManifest*.xml file.

Let's look at an example of how to enable an application to take input in the form of a phone number and launch the Phone dialer after the user presses a button, as shown in Figure 13.6.

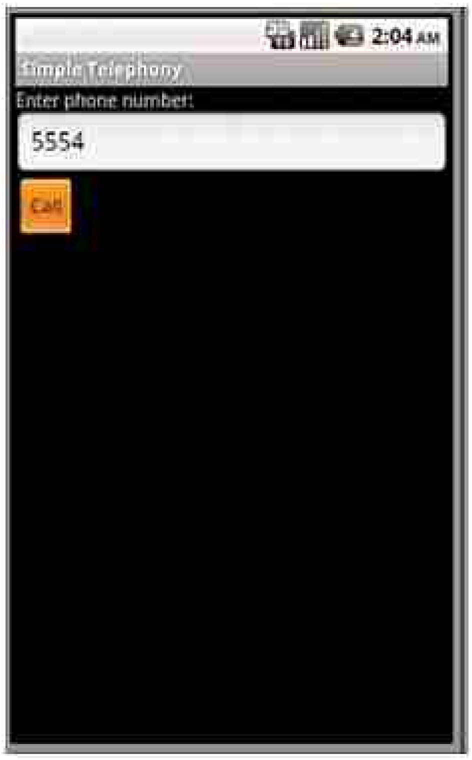


**Figure 3.6 : The user can enter a phone number in the EditText control and press the Call button to initiate a phone call from within the application.**

We extract the phone number the user entered in the EditText field (or the most recently received SMS when continuing with the previous example).

The following code demonstrates how to launch the dialer after the user presses the Call button :

```
Button call = (Button) findViewById(R.id.call_button);
call.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
Uri number = Uri.parse("tel:" +
numberEntry.getText().toString());
Intent dial = new Intent(
Intent.ACTION_DIAL, number);
startActivity(dial);
}
});
```

First, the phone number is requested from the EditText and tel: is prepended to it, making it a valid Uri for the Intent.Then, a new Intent is created with Intent.ACTION_DIAL to launch in to the dialer with the number dialed in already. You can also use Intent.ACTION_VIEW, which functions the same. Replacing it with Intent.ACTION_CALL, however, immediately calls the number entered.This is generally not recommended; otherwise, calls might be made by mistake. Finally, the startActivity() method is called to launch the dialer, as shown in Figure 13.7

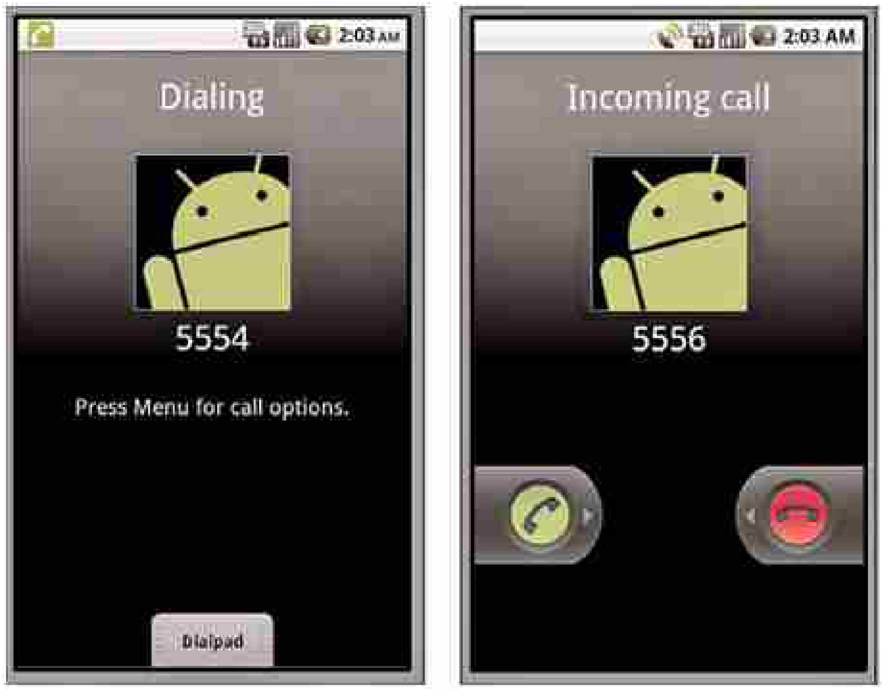


**Figure 13.7 : One emulator calling the other after the Call button is pressed within the application.**

**Receiving Phone Calls**

Much like applications can receive and process incoming SMS messages, an application can register to answer incoming phone calls. To enable this within an application, you must implement a broadcast receiver to process intents with the action Intent.ACTION_ANSWER. Remember, too, that if you're not interested in the call itself, but information about the incoming call, you might want to consider using the CallLog.Calls content provider (android.provider.CallLog) instead. You can use the CallLog.calls class to determine recent call information, such as

- Who called
- When they called
- Whether it was an incoming or outgoing call
- Whether or not anyone answered
- The duration of the call

❑ **Check Your Progress :**

1. ________ control uses the WebKit rendering engine to draw HTML content on the screen.

   (A) The WebView (B) WebKit

   (C) Web (D) View

2. ________ class enables the application to listen for certain WebView events.

   (A) WebKit (B) WebViewClient

   (C) The WebView (D) View

3. What is specialized for the sorts of items that will be drawn outside the region in which the web content is drawn ?

   (A) WebViewClient (B) The WebView

   (C) WebChromeClient (D) WebKit

4. Which object is used to retrieve the state of the phone and some information about the phone service itself ?

   (A) ConnectionManager (B) SmsManager

   (C) ContentManager (D) TelephonyManager

5. To send an SMS, an application first needs to get an instance of the ________

   (A) SmsManager (B) ConnectionManager

   (C) TelephonyManager (D) ConnectionManager

## 13.7 Let Us Sum Up :

In this unit, we discussed regarding to learn how to use Web API in Android Application, to learn how to Design Layout with a WebView Control, to understand how to apply WebKit API , to learn and create Flash Application Using Android, to lean about Telephony API in Android Application, to understand and create mobile application with calling a number and to understand and create mobile application to send the message on number

## 13.8 Answers for Check Your Progress :

**1.** (C) **2.** (A) **3.** (C) **4.** (D) **5.** (A)

## 13.9 Glossary :

1. **SDK :** Software Development Kit
2. **AVD :** Android Virtual Device
3. **DDMS :** Dalvik Debug Monitor Server
4. **ADB :** Android Debug Bridge

## 13.10 Assignment :

1. What is the use of Web API in android ?
2. Explain basic classes and methods available in Web API.
3. How to call and send a SMS to particular number ?
4. Explain Android WebView control in details.
5. How to load content in WebView Control ?
6. Explain basic classes and methods available in Telephony API.

## 13.11 Activities :

1. Identify the real–life application which use the web browser to access the internet.
2. Identify the real–life application which use to call and send SMS to a specified given number for call and SMS.

## 13.12 Case Study :

Create the real–life application which use the web browser to access the internet.

Create the real–life application which use to call and send SMS to a specified given number for call and SMS.

## 13.13 Further Reading :

Lauren Darcey and Shane Conder, "Android Wireless Application Development", Pearson Education, 2nd ed. (2011).

Unit 14

# SELLING YOUR ANDROID APPLICATION

**UNIT STRUCTURE**

## 14.0 Learning Objectives :

- To learn how to choose right distribution model
- To learn how to package Android application for Publication
- To understand how to distribute the Android Application

## 14.1 Introduction :

After you've developed an application, the next logical step is to publish it so that other people can enjoy it. You might even want to make some money. There are a variety of distribution opportunities available to Android application developers. Many developers choose to sell their applications through mobile marketplaces such as Google's Android Market. Others develop their own distribution mechanisms–for example, they might sell their applications from a website. Regardless, developers should consider which distribution options they plan to use during the application design and development process, as some distribution choices might require code changes or impose restrictions on content.

## 14.2 Choosing the Right Distribution Model :

The application distribution methods you choose to employ depend on your goals and target users. Some questions you should ask yourself are n Is your application ready for prime time or are you considering a beta period to iron out the kinks ?

- Are you trying to reach the broadest audience, or have you developed a vertical market application ? Determine who your users are, which devices they are using, and their preferred methods for seeking out and downloading applications.
- How will you price your application ? Is it freeware or shareware ? Are the payment models (single payment versus subscription model versus ad–driven revenue) you require available on the distribution mechanisms you want to leverage?
- Where do you plan to distribute ? Verify that any application markets you plan to use are capable of distributing within those countries or regions.
- Are you willing to share a portion of your profits ? Distribution mechanisms such as the Android Market take a percentage of each sale in exchange for hosting your application for distribution and collecting application revenue on your behalf.
- Do you require complete control over the distribution process or are you willing to work within the boundaries and requirements imposed by third–party marketplaces? This might require compliance with further license agreements and terms.
- If you plan to distribute yourself, how will you do so ? You might need to develop more services to manage users, deploy applications and collect payments. If so, how will you protect user data ? What trade laws must you comply with ?
- Have you considered creating a free trial version of your application? If the distribution system under consideration has a return policy, consider the ramifications. You need to ensure that your application has safeguards to minimize the number of users that buy your app, use it, and return it for a full refund. For example, a game might include safeguards such as a free trial version and a full–scale version with more game levels than could possibly be completed within the refundable time period.

Now let's look at the steps you need to take to package and publish your application.

## 14.3 Packaging Your Application for Publication :

There are several steps developers must take when preparing an Android application for publication and distribution. Your application must also meet several important requirements imposed by the marketplaces. The following steps are required for publishing an application :

1. Prepare and perform a release candidate build of the application.
2. Verify that all requirements for marketplace are met, such as configuring the Android manifest file properly. For example, make sure the application name and version information is correct and the debuggable attribute is set to false.
3. Package and digitally sign the application.
4. Test the packaged application release thoroughly.
5. Publish the application.

The preceding steps are required but not sufficient to guarantee a successful deployment.

Developers should also

1. Thoroughly test the application on all target handsets.
2. Turn off debugging, including Log statements and any other logging.
3. Verify permissions, making sure to add ones for services used and remove any that aren't used, regardless of whether they are enforced by the handsets.
4. Test the final, signed version with all debugging and logging turned off.

Now, let's explore each of these steps in more detail, in the order they might be performed.

### 14.3.1 Preparing Your Code to Package :

An application that has undergone a thorough testing cycle might need changes made to it before it is ready for a production release. These changes convert it from a debuggable, preproduction application into a release–ready application.

**Setting the Application Name and Icon**

An Android application has default settings for the icon and label. The icon appears in the application Launcher and can appear in various other locations, including marketplaces. As such, an application is required to have an icon. You should supply alternate icon drawable resources for various screen resolutions. The label, or application name, is also displayed in similar locations and defaults to the package name. You should choose a user–friendly name.

**Versioning the Application**

Next, proper versioning is required, especially if updates could occur in the future. The version name is up to the developer. The version code, though, is used internally by the Android system to determine if an application is an update. You should increment the version code for each new update of an application. The exact value doesn't matter, but it must be greater than the previous version code. Versioning within the Android manifest file is discussed in previous chapter.

**Verifying the Target Platforms**

Make sure your application sets the <uses–sdk> tag in the Android manifest file correctly. This tag is used to specify the minimum and target platform versions that the application can run on. This is perhaps the most important setting after the application name and version information.

**Configuring the Android Manifest for Market Filtering**

If you plan to publish through the Android Market, you should read up on how this distribution system uses certain tags within the Android manifest file to filter applications available to users. Many of these tags, such as <supports–screens>, <uses–configuration>, <uses–feature>, <uses–library>, <uses–permission>, and <uses–sdk>, were discussed in previous chapter. Set each of these settings carefully, as you don't want to accidentally put too many restrictions on your application. Make sure you test your application thoroughly after configuring these Android manifest file settings. For more information on how Android Market filters work, see http://developer.android.com/guide/ appendix/ market–filters.html.

**Preparing Your Application Package for the Android Market**

The Android Market has strict requirements on application packages. When you upload your application to the Android Market website, the package is verified and any problems are communicated to you. Most often, problems occur when you have not properly configured your Android manifest file.

The Android Market uses the android:versionName attribute of the <manifest> tag within the Android manifest file to display version information to users. It also uses the android:versionCode attribute internally to handle application upgrades. The android:icon and android:label attributes must also be present because both are used by the Android Market to display the application name to the user with a visual icon.

**Disabling Debugging and Logging**

Next, you should turn off debugging and logging. Disabling debugging involves removing the android:debuggable attribute from the <application> tag of the AndroidManifest.xml file or setting it to false. You can turn off the logging code within Java in a variety of different ways, from just commenting it out to using a build system that can do this automatically.

**Verifying Application Permissions**

Finally, the permissions used by the application should be reviewed. Include all permissions that the application requires, and remove any that are not used. Users appreciate this.

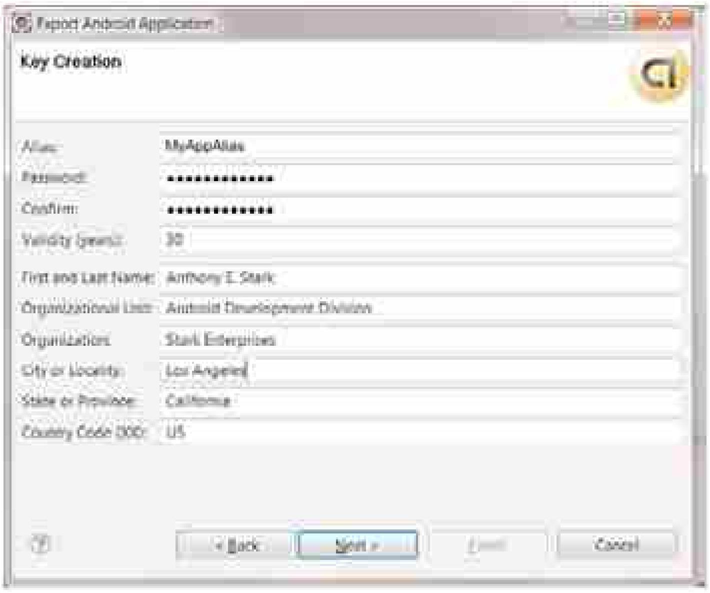
### 14.3.2 Packing and Signing Your Application :

Now that the application is ready for publication, the file package–the .apk file–needs to be prepared for release. The package manager of an Android device will not install a package that has not been digitally signed. Throughout the development process, the Android tools have accomplished this through signing with a debug key. The debug key cannot be used for publishing an application to the wider world. Instead, you need to use a true key to digitally sign the application. You can use the private key to digitally sign the release package files of your Android application, as well as any upgrades. This ensures that the application (as a complete entity) is coming from you, the developer, and not some other source (imposters!).

The Android Market requires that your application's digital signature validity period end after October 22, 2033. This date might seem like a long way off and, for mobile, it certainly is. However, because an application must use the same key for upgrading and applications that want to work closely together with special privileges and trust relationships must also be signed with the same key, the key could be chained forward through many applications. Thus, Google is mandating that the key be valid for the foreseeable future so application updates and upgrades are performed smoothly for users. Although self–signing is typical of Android applications, and a certificate authority is not required, creating a suitable key and securing it properly is critical. The digital signature for Android applications can impact certain functionality. The expiry of the signature is verified at installation time, but after it's installed, an application continues to function even if the signature has expired.

You can export and sign your Android package file from within Eclipse using the Android Development plug–in, or you can use the command–line tools. You can export and sign your Android package file from within Eclipse by taking the following steps :

1. In Eclipse, right–click the appropriate application project and choose the Export option.
2. Under the Export menu, expand the Android section and choose Export Android Application.
3. Click the Next button.
4. Select the project to export (the one you right–clicked before is the default).
5. On the keystore selection screen, choose the Create New Keystore option and enter a file location (where you want to store the key) as well as a password for managing the keystore. (If you already have a keystore, choose browse to pick your keystore file, and then enter the correct password.)
6. Click the Next button.
7. On the Key Creation screen, enter the details of the key, as shown in Figure 14.1.



| | |
|---|---|
| Alias: | MyAppAlias |
| Password: | ••••••••••••• |
| Confirm: | ••••••••••••• |
| Validity (years): | 30 |
| First and Last Name: | Anthony E. Stark |
| Organizational Unit: | Android Development Division |
| Organization: | Stark Enterprises |
| City or Locality: | Los Angeles |
| State or Province: | California |
| Country Code (XX): | US |

**Figure 14.1 : Exporting and signing an Android application in Eclipse.**

8. Click the Next button.
9. On the Destination and Key/Certificate Checks screen, enter a destination for the application package file.
10. Click the Finish button.

You have now created a fully signed and certified application package file. The application package is ready for publication.

**Testing the Release Version of Your Application Package**

Now that you have configured your application for production, you should perform a full final testing cycle paying special attention to subtle changes to the installation process. An important part of this process is to verify that you have disabled all debugging features and logging has no negative impact on the functionality and performance of the application.

**Certifying Your Android Application**

If you're familiar with other mobile platforms, you might be familiar with the many strict certification programs found on platforms, such as the TRUE BREW or Symbian Signed programs. These programs exist to enforce a lower bound on the quality of an application.

As of this writing, Android does not have any certification or testing requirements. It is an open market with only a few content guidelines and rules to follow. This does not mean, however, that certification won't be required at some point or that certain distribution means won't require certification.

Typically, certification programs require rigorous and thorough testing, certain usability conventions must be met, and various other constraints that might be good common practice or operator–specific rules are enforced. The best way to prepare for any certification program is to incorporate its requirements into the design of your specific project.

Following best practices for Android development and developing efficient, usable, dynamic, and robust applications always pay off in the end–whether your application requires certification.

## 14.4 Distributing Your Applications :

Now that you've prepared your application for publication, it's time to get your application out to users–for fun and profit. Unlike other mobile platforms, most Android distribution mechanisms support free applications and price plans.

### 14.4.1 Selling Your Application on the Android Market :

The Android Market is the primary mechanism for distributing Android applications at this time. This is where your typical user purchases and downloads applications. As of this writing, it's available to most, but not all, Android devices. As such, we show you how to check your package for preparedness, sign up for a developer account, and submit your application for sale on the Android Market.

### 14.4.2 Signing Up for a Developer Account on the Android Market :

To publish applications through the Android Market, you must register as a developer. This accomplishes two things. It verifies who you are to Google and signs you up for a Google Checkout account, which is used for billing of Android applications.

To sign up for an Android Market developer account, you need to follow these steps :

1. Go to the Android Market sign–up website at http://market.android.com/ publish/ signup, as shown in Figure 14.2.
2. Sign in with the Google Account you want to use. (At this time, you cannot change the associated Google Account, but you can change the contact email addresses for applications independently.)
3. Enter your developer information, including your name, email address, and website, as shown in Figure 14.3.

4. Confirm your registration payment (as of this writing, $25 USD). Note that Google Checkout is used for registration payment processing.
5. Signing up and paying to be an Android Developer also creates a mandatory Google Checkout Merchant account for which you also need to provide information. This account is used for payment processing purposes.
6. Agree to link your credit card and account registration to the Android Market Developer Distribution Agreement. The basic agreement (U.S. version) is available for review at http://www.android.com/us/developer–distribution–agreement.html.

Always print out the actual agreement you sign as part of the registration process, in case it changes in the future.

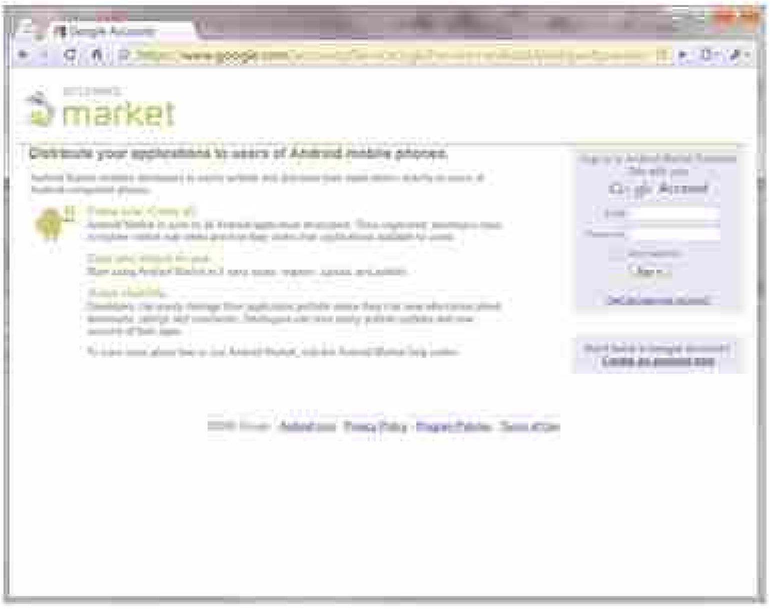

market
Distribute your applications to users of Android mobile phones.
Google Account

**Figure 14.2 : The Android Market publisher sign–up page.**

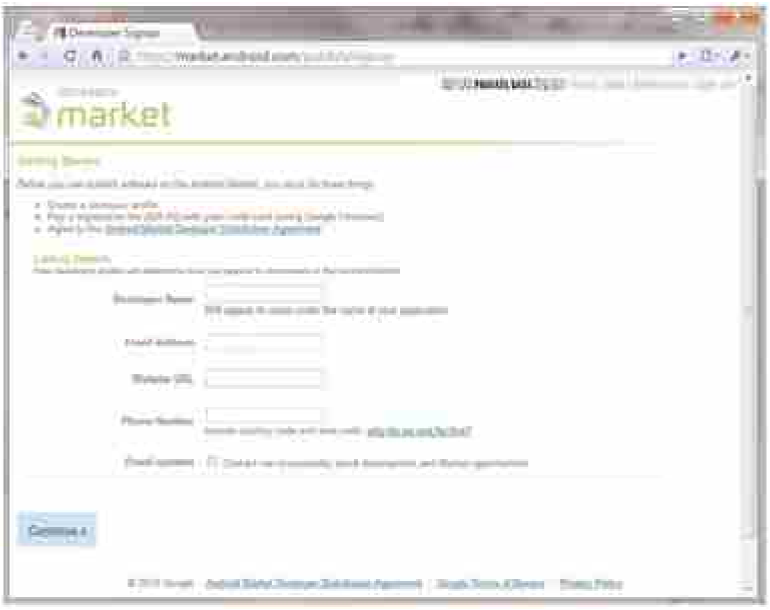


**Figure 14.3 : The Android Market publisher profile page.**

When you successfully complete these steps, you are presented with the home screen of the Android Market, which also confirms that the Google Checkout Merchant account was created.

**Uploading Your Application to the Android Market**

Now that you have an account registered for publishing applications through Android Market and a signed application package, you are ready to upload it for publication. From the main page of the Android Market website (http://market.android.com/publish), sign in with your developer account information. After you are logged in, you see a webpage with your developer account information, as shown in Figure 14.4.

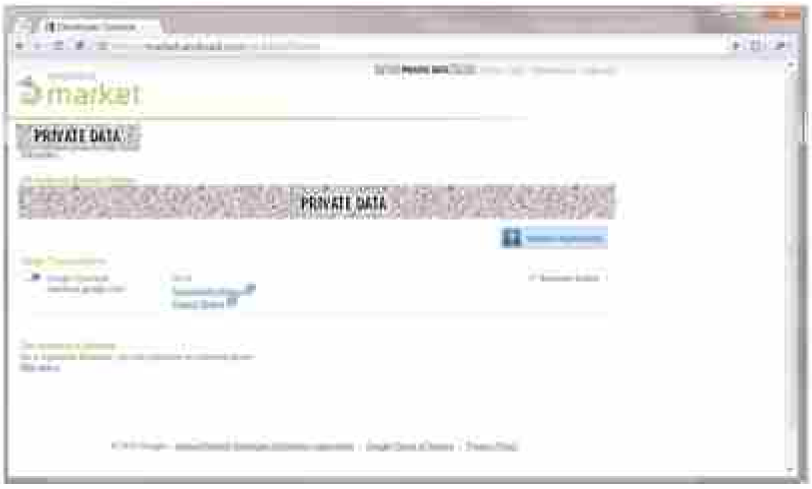


**Figure 14.4 : Android Market developer application listings.**

From this page, you can configure developer account settings, see your payment transaction history, and manage your published applications. In order to publish a new application, press the Upload Application button on this page.A form is presented for uploading the application package (see Figure 14.5).

Let's look at some of the important fields you must enter on this form :

- The application package file (.apk)
- Promotional screenshots and graphics for the market listing
- The application title and description in several languages
- Application type and category

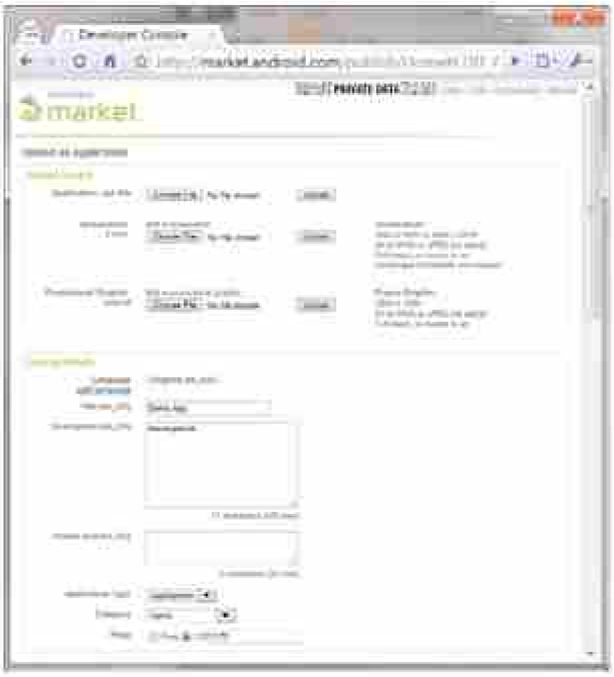


**Figure 14.5 : Android Market application upload form.**

- Application price–Free or Paid (this cannot be changed later)
- Copy protection information–Choosing this option might help prevent the application from being copied from the device and distributed without your knowledge or permission. This option is likely to change in the near future, as Google adds a new licensing service.
- Locations to distribute to–Choose the countries where the application should be published.
- Support contact information–This option defaults to the information you provided for the developer account. You can change it on an app–by–app basis, though, which allows for great support flexibility when you're publishing multiple applications.
- Consent–You must click the checkboxes to agree to the terms of the current (at the time you click) Android Content Guidelines, as well as the export laws of the United States, regardless of your location or nationality.

**Publishing Your Application on the Android Market**

Finally, you are ready to press the Publish button. Your application appears in the Android Market almost immediately. After publication, you can see statistics including ratings, reviews, downloads, and active installs in the Your Android Market Listings section of the main page on your developer account. These statistics aren't updated as frequently as the publish action is, and you can't see review details directly from the listing. Clicking on the application listing enables you to edit the various fields.

**Understanding the Android Market Application Return Policy**

Although it is a matter of no small controversy, the Android Market has a 24–hour refund policy on applications. That is to say, a user can use an application for 24 hours and then return it for a full refund. As a developer, this means that sales aren't final until after the first 24 hours. However, this only applies to the first download and first return. If a particular user has already returned your application and wants to "try it again," he or she must make a final purchase–and can't return it a second time. Although this limits abuse, you should still be aware that if your application has limited reuse appeal or if all its value can come from just a few hours (or less) of use, you might find that you have a return rate that's too high and need to pursue other methods of monetization.

**Upgrading Your Application on the Android Market**

You can upgrade existing applications from the Market from the developer account page. Simply upload a new version of the same application using the Android manifest file tag, android:versionCode. When you publish it, users receive an Update Available notification, prompting them to download the upgrade.

**Removing Your Application from the Android Market**

You can also use the unpublish action to remove the application from the Market from the developer account. The unpublish action is also immediate, but the application entry on the Market application might be cached on handsets that have viewed or downloaded the application.

### Using Other Developer Account Benefits

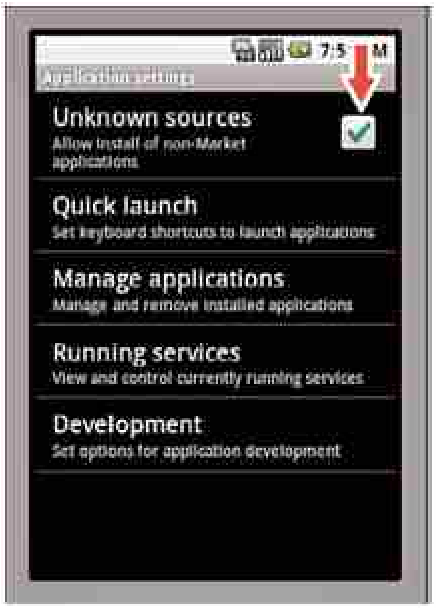
In addition to managing your applications on the Android Market, an additional benefit to have a registered Android developer account is the ability to purchase development versions of Android handsets. These handsets are useful for general development and testing but might not be suitable for final testing on actual target handsets because some functionality might be limited, and the firmware version might be different than that found on consumer handsets.

### Selling Your Application on Your Own Server

You can distribute Android applications directly from a website or server. This method is most appropriate for vertical market applications, content companies developing mobile marketplaces, and big brand websites wanting to drive users to their branded Android applications. It can also be a good way to get beta feedback from end users.

Although self–distribution is perhaps the easiest method of application distribution, it might also be the hardest to market, protect, and make money. The only requirement for self–distribution is to have a place to host the application package file.

The downside of self–distribution is that end users must configure their devices to allow packages from unknown sources. This setting is found under the Applications section of the device Settings application, as shown in Figure 14.6. This option is not available on all consumer devices in the market. Most notably, Android devices on U.S. carrier AT&T can only install applications from the Android Market–no third–party sources are allowed.



Application settings

Unknown sources ✓
Allow install of non-Market applications

Quick launch
Set keyboard shortcuts to launch applications

Manage applications
Manage and remove installed applications

Running services
View and control currently running services

Development
Set options for application development

**Figure 14.6 : Settings application showing required check box for downloading from unknown sources.**

After that, the final step the user must make is to enter the URL of the application package in to the web browser on the handset and download the file (or click on a link to it). When downloaded, the standard Android install process occurs, asking the user to confirm the permissions and, optionally, confirm an update or replacement of an existing application if a version is already installed.

❑ **Check Your Progress :**

1. The Android Market uses the ________ attribute of the <manifest> tag within the Android manifest file to display version information to users.

   (A) android:versionName (B) android:versionCode

   (C) android:version (D) android:dibuggable

2. Which attribute is used to enable and disable the debugging option ?

   (A) android:versionName (B) android:versionCode

   (C) android:version (D) android:dibuggable

3. Give the name of the attribute which used to set while we upgrade the android application in manifest file.

   (A) android:versionName (B) android:versionCode

   (C) android:version (D) android:dibuggable

4. On which of the following, developers can test the application, during developing the android applications ?

   (A) Third–party emulators

   (B) Emulator included in Android SDK

   (C) Physical android phone

   (D) All of the above

5. As an android programmer, which version of Android should we use as a minimum development target ?

   (A) Version 1.2 or version 1.3 (B) Version 1.0 or version 1.1

   (C) Version 1.6 or version 2.0 (D) Version 2.3 or version 3.0

## 14.5 Let Us Sum Up :

In this unit, we discussed regarding to learn how to choose right distribution model, to learn how to package Android application for Publication and to understand how to distribute the Android Application

## 14.6 Answers for Check Your Progress :

**1.** (A) **2.** (D) **3.** (B) **4.** (D) **5.** (C)

## 14.7 Glossary :

**1.** **SDK :** Software Development Kit

**2.** **AVD :** Android Virtual Device

**3.** **DDMS :** Dalvik Debug Monitor Server

**4.** **ADB :** Android Debug Bridge

## 14.8 Assignment :

1. Which steps are required for publishing an Android application ?
2. List out the steps to Package Your Application for Publication.
3. Explain in details the steps require to Upload Your Application to the Android Market.
4. What is the role of AndroidManifest.xml file in Android Application ?
5. Discuss how to remove your application from Android Market.

## 14.9 Activities :

1. Try the application to package, publish and upload in Android Market for selling.

## 14.10 Case Study :

Package, upload and publish the developed application in Android Market to sell the application.

## 14.11 Further Reading :

Lauren Darcey and Shane Conder, "Android Wireless Application Development", Pearson Education, 2nd ed. (2011).

## BLOCK SUMMARY :

In this block we learnt regarding to understand the common API available in Android, to implement Database Connection Application using SQLite Database, to understand the network and web connection using implementation of Network and Web APIs, to create the Android Application for Calling & Message sending using Android Telephony API and to understand how to deploy Android and sell the Android Application

## BLOCK ASSIGNMENT :

1. Explain how to Create, Update, and Delete Database Records in SQLite with example.
2. What is Cursor? Explain the use of Cursor in SQLite Database.
3. Explain the use of ContentValues to insert the record and Which four arguments are taken by update() ?
4. What is SQLite ?
5. How to create SQLite database ?
6. How to create tables in SQLite database ?
7. What is Android networking ?
8. What are network libraries in Android ?
9. What is networking on phone ?
10. Explain in details Android Networking API with its supported methods.
11. What is the use of Web API in android ?
12. Explain basic classes and methods available in Web API.
13. How to call and send a SMS to particular number ?
14. Explain Android WebView control in details.
15. How to load content in WebView Control ?
16. Explain basic classes and methods available in Telephony API
17. Which steps are required for publishing an Android application ?
18. List out the steps to Package Your Application for Publication.
19. Explain in details the steps require to Upload Your Application to the Android Market.
20. What is the role of AndroidManifest.xml file in Android Application ?
21. Discuss how to remove your application from Android Market.

❖ **Short Questions :**

1. What is SQLite ?
2. How to create SQLite database ?
3. How to create tables in SQLite database ?
4. What is Android networking ?
5. What is networking on phone ?
6. How to load content in WebView Control ?
7. How to call and send a SMS to particular number ?
8. List out the steps to Package Your Application for Publication

❖ **Long Questions :**

1. Explain how to Create, Update, and Delete Database Records in SQLite with example.
2. What is Cursor ? Explain the use of Cursor in SQLite Database.
3. Explain the use of ContentValues to insert the record and Which four arguments are taken by update() ?
4. Explain in details Android Networking API with its supported methods.
5. What is the use of Web API in android ?
6. Explain basic classes and methods available in Web API.
7. How to call and send a SMS to particular number ?
8. Explain basic classes and methods available in Web API.
9. Explain Android WebView control in details.
10. Explain basic classes and methods available in Telephony API
11. Which steps are required for publishing an Android application ?
12. List out the steps to Package Your Application for Publication.
13. Explain in details the steps require to Upload Your Application to the Android Market.
14. What is the role of AndroidManifest.xml file in Android Application ?
15. Discuss how to remove your application from Android Market.

❖ **Enrolment No. :** ☐

1. How many hours did you need for studying the units ?

| Unit No. | 11 | 12 | 13 | 14 |
|---|---|---|---|---|
| No. of Hrs. | | | | |

2. Please give your reactions to the following items based on your reading of the block :

| Items | Excellent | Very Good | Good | Poor | Give specific example if any |
|---|---|---|---|---|---|
| Presentation Quality | ☐ | ☐ | ☐ | ☐ | ________ |
| Language and Style | ☐ | ☐ | ☐ | ☐ | ________ |
| Illustration used (Diagram, tables etc) | ☐ | ☐ | ☐ | ☐ | ________ |
| Conceptual Clarity | ☐ | ☐ | ☐ | ☐ | ________ |
| Check your progress Quest | ☐ | ☐ | ☐ | ☐ | ________ |
| Feed back to CYP Question | ☐ | ☐ | ☐ | ☐ | ________ |

3. Any other Comments

..........................................................................................................

..........................................................................................................

..........................................................................................................

..........................................................................................................

..........................................................................................................

..........................................................................................................

..........................................................................................................

..........................................................................................................