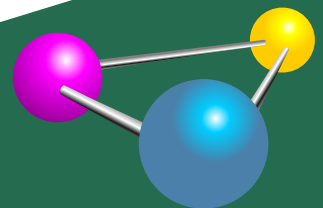


**Version
2.1**

*Supports the
Android 1.5
SDK!*

The Busy Coder's Guide to Android Development

Mark L. Murphy



COMMONSWARE

The Busy Coder's Guide to Android Development

by Mark L. Murphy

The Busy Coder's Guide to Android Development

by Mark L. Murphy

Copyright © 2008-2009 CommonsWare, LLC. All Rights Reserved.
Printed in the United States of America.

CommonsWare books may be purchased in printed (bulk) or digital form for educational or business use. For more information, contact *direct@commonsware.com*.

Printing History:

Jun 2009:

Version 2.1

ISBN: 978-0-9816780-0-9

The CommonsWare name and logo, "Busy Coder's Guide", and related trade dress are trademarks of CommonsWare, LLC.

All other trademarks referenced in this book are trademarks of their respective firms.

The publisher and author(s) assume no responsibility for errors or omissions or for damages resulting from the use of the information contained herein.

The Busy Coder's Guide to Android Development

by Mark L. Murphy

Copyright © 2008-2009 CommonsWare, LLC. All Rights Reserved.
Printed in the United States of America.

CommonsWare books may be purchased in printed (bulk) or digital form for educational or business use. For more information, contact *direct@commonsware.com*.

Printing History:

Jun 2009:

Version 2.1

ISBN: 978-0-9816780-0-9

The CommonsWare name and logo, "Busy Coder's Guide", and related trade dress are trademarks of CommonsWare, LLC.

All other trademarks referenced in this book are trademarks of their respective firms.

The publisher and author(s) assume no responsibility for errors or omissions or for damages resulting from the use of the information contained herein.

Table of Contents

Welcome to the Warescription!	xv
Preface	xvii
Welcome to the Book!.....	xvii
Prerequisites.....	xvii
Warescription.....	xviii
Book Bug Bounty.....	xix
Source Code License.....	xx
Creative Commons and the Four-to-Free (42F) Guarantee.....	xx
Acknowledgments.....	xxi
The Big Picture	1
What Androids Are Made Of.....	3
Activities.....	3
Content Providers.....	4
Intents.....	4
Services.....	4
Stuff At Your Disposal.....	4
Storage.....	4
Network.....	5
Multimedia.....	5

GPS.....	5
Phone Services.....	5
Project Structure.....	7
Root Contents.....	7
The Sweat Off Your Brow.....	8
And Now, The Rest of the Story.....	8
What You Get Out Of It.....	9
Inside the Manifest.....	11
In The Beginning, There Was the Root, And It Was Good.....	11
Permissions, Instrumentations, and Applications (Oh, My!).....	12
Your Application Does Something, Right?.....	13
Achieving the Minimum.....	14
Version=Control.....	16
Getting Going.....	17
Virtually There.....	17
Aiming at a Target.....	18
Creating a Skeleton Application.....	23
Begin at the Beginning.....	23
The Activity.....	24
Dissecting the Activity.....	25
Building and Running the Activity.....	27
Using XML-Based Layouts.....	31
What Is an XML-Based Layout?.....	31
Why Use XML-Based Layouts?.....	32
OK, So What Does It Look Like?.....	33
What's With the @ Signs?.....	34
And We Attach These to the Java...How?.....	34

The Rest of the Story.....	35
Employing Basic Widgets.....	39
Assigning Labels.....	39
Button, Button, Who's Got the Button?.....	40
Fleeting Images.....	41
Fields of Green. Or Other Colors.....	42
Just Another Box to Check.....	45
Turn the Radio Up.....	48
It's Quite a View.....	50
Useful Properties.....	50
Useful Methods.....	50
Colors.....	51
Working with Containers.....	53
Thinking Linearly.....	54
Concepts and Properties.....	54
Example.....	57
All Things Are Relative.....	62
Concepts and Properties.....	62
Example.....	65
Tabula Rasa.....	68
Concepts and Properties.....	68
Example.....	71
Scrollwork.....	72
Using Selection Widgets.....	77
Adapting to the Circumstances.....	77
Using ArrayAdapter.....	78
Other Key Adapters.....	79

Lists of Naughty and Nice.....	80
Selection Modes.....	82
Spin Control.....	84
Grid Your Lions (Or Something Like That.....)	88
Fields: Now With 35% Less Typing!.....	92
Galleries, Give Or Take The Art.....	96
Getting Fancy With Lists.....	99
Getting To First Base.....	99
A Dynamic Presentation.....	102
A Sidebar About Inflation.....	103
And Now, Back To Our Story.....	104
Better. Stronger. Faster.....	105
Using convertView.....	106
Using the Holder Pattern.....	108
Making a List.....	111
...And Checking It Twice.....	117
Adapting Other Adapters.....	124
Employing Fancy Widgets and Containers.....	125
Pick and Choose.....	125
Time Keeps Flowing Like a River.....	130
Making Progress.....	131
Putting It On My Tab.....	132
The Pieces.....	133
The Idiosyncrasies.....	133
Wiring It Together.....	135
Adding Them Up.....	138
Intents and Views.....	141

Flipping Them Off.....	141
Getting In Somebody's Drawer.....	147
Other Good Stuff.....	150
The Input Method Framework.....	153
Keyboards, Hard and Soft.....	153
Tailored To Your Needs.....	154
Tell Android Where It Can Go.....	158
Fitting In.....	160
Unleash Your Inner Dvorak.....	162
Applying Menus.....	163
Flavors of Menu.....	163
Menus of Options.....	164
Menus in Context.....	166
Taking a Peek.....	167
Yet More Inflation.....	173
Menu XML Structure.....	173
Menu Options and XML.....	175
Inflating the Menu.....	176
Fonts.....	179
Love The One You're With.....	179
Here a Glyph, There a Glyph.....	182
Embedding the WebKit Browser.....	185
A Browser, Writ Small.....	185
Loading It Up.....	188
Navigating the Waters.....	189
Entertaining the Client.....	190
Settings, Preferences, and Options (Oh, My!).....	192

Showing Pop-Up Messages	195
Raising Toasts.....	195
Alert! Alert!.....	196
Checking Them Out.....	197
Dealing with Threads	201
Getting Through the Handlers.....	201
Messages.....	202
Runnables.....	205
Running In Place.....	206
Where, Oh Where Has My UI Thread Gone?.....	206
Asyncing Situation.....	206
The Theory.....	206
AsyncTask, Generics, and Varargs.....	207
The Stages of AsyncTask.....	208
A Sample Task.....	209
And Now, The Caveats.....	214
Handling Activity Lifecycle Events	215
Schroedinger's Activity.....	215
Life, Death, and Your Activity.....	216
onCreate() and onDestroy().....	216
onStart(), onRestart(), and onStop().....	217
onPause() and onResume().....	218
The Grace of State.....	218
Using Preferences	223
Getting What You Want.....	223
Stating Your Preference.....	224
And Now, a Word From Our Framework.....	225

Letting Users Have Their Say.....	226
Adding a Wee Bit O' Structure.....	231
The Kind Of Pop-Ups You Like.....	234
Accessing Files.....	239
You And The Horse You Rode In On.....	239
Readin' 'n Writin'.....	243
Working with Resources.....	249
The Resource Lineup.....	249
String Theory.....	250
Plain Strings.....	250
String Formats.....	251
Styled Text.....	251
Styled Formats.....	252
Got the Picture?.....	256
XML: The Resource Way.....	259
Miscellaneous Values.....	261
Dimensions.....	262
Colors.....	263
Arrays.....	263
Different Strokes for Different Folks.....	264
Managing and Accessing Local Databases.....	271
A Quick SQLite Primer.....	272
Start at the Beginning.....	273
Setting the Table.....	274
Makin' Data.....	275
What Goes Around, Comes Around.....	276
Raw Queries.....	276

Regular Queries.....	277
Building with Builders.....	278
Using Cursors.....	279
Making Your Own Cursors.....	280
Data, Data, Everywhere.....	280
Leveraging Java Libraries.....	283
The Outer Limits.....	283
Ants and Jars.....	284
Following the Script.....	285
...And Not A Drop To Drink.....	290
Reviewing the Script.....	291
Communicating via the Internet.....	293
REST and Relaxation.....	293
HTTP Operations via Apache HttpComponents.....	294
Parsing Responses.....	296
Stuff To Consider.....	298
Creating Intent Filters.....	303
What's Your Intent?.....	304
Pieces of Intents.....	304
Intent Routing.....	305
Stating Your Intent(ions).....	306
Narrow Receivers.....	308
The Pause Caveat.....	309
Launching Activities and Sub-Activities.....	311
Peers and Subs.....	312
Start 'Em Up.....	312
Make an Intent.....	313

Welcome to the Book!

Thanks!

Thanks for your interest in developing applications for Android! Increasingly, people will access Internet-based services using so-called "non-traditional" means, such as mobile devices. The more we do in that space now, the more that people will help invest in that space to make it easier to build more powerful mobile applications in the future. Android is new – Android-powered devices appeared on the scene first in late 2008 – but it likely will rapidly grow in importance due to the size and scope of the Open Handset Alliance.

And, most of all, thanks for your interest in this book! I sincerely hope you find it useful and at least occasionally entertaining.

Prerequisites

If you are interested in programming for Android, you will need at least basic understanding of how to program in Java. Android programming is done using Java syntax, plus a class library that resembles a subset of the Java SE library (plus Android-specific extensions). If you have not programmed in Java before, you probably should learn how that works before attempting to dive into programming for Android.

The book does not cover in any detail how to download or install the Android development tools, either the Eclipse IDE flavor or the standalone flavor. The [Android Web site](#) covers this quite nicely. The material in the book should be relevant whether you use the IDE or not. You should download, install, and test out the Android development tools from the Android Web site before trying any of the examples listed in this book.

Some chapters may reference material in previous chapters, though usually with a link back to the preceding section of relevance. Also, not every sample shown has the complete source code in the book, lest this book get too large. If you wish to compile the samples, [download the source code](#) from the CommonsWare Web site.

Warescription

This book will be published both in print and in digital (ebook) form. The ebook versions of all CommonsWare titles are available via an annual subscription – the Warescription.

The Warescription entitles you, for the duration of your subscription, to ebook forms of *all* CommonsWare titles, not just the one you are reading. Presently, CommonsWare offers PDF and Kindle; other ebook formats will be added based on interest and the openness of the format.

Each subscriber gets personalized editions of all editions of each title: both those mirroring printed editions and in-between updates that are only available in ebook form. That way, your ebooks are never out of date for long, and you can take advantage of new material as it is made available instead of having to wait for a whole new print edition. For example, when new releases of the Android SDK are made available, this book will be quickly updated to be accurate with changes in the APIs.

From time to time, subscribers will also receive access to subscriber-only online material, both short articles and not-yet-published new titles.

Also, if you own a print copy of a CommonsWare book, and it is in good clean condition with no marks or stickers, you can exchange that copy for a discount off the Warescription price.

If you are interested in a Warescription, visit the Warescription section of the CommonsWare [Web site](#).

Book Bug Bounty

Find a problem in one of our books? Let us know!

Be the first to report a unique concrete problem in the current digital edition, and we'll give you a coupon for a six-month Warescription as a bounty for helping us deliver a better product. You can use that coupon to get a new Warescription, renew an existing Warescription, or give the coupon to a friend, colleague, or some random person you meet on the subway.

By "concrete" problem, we mean things like:

- Typographical errors
- Sample applications that do not work as advertised, in the environment described in the book
- Factual errors that cannot be open to interpretation

By "unique", we mean ones not yet reported. Each book has an errata page on the CommonsWare Web site; most known problems will be listed there. One coupon is given per email containing valid bug reports.

We appreciate hearing about "softer" issues as well, such as:

- Places where you think we are in error, but where we feel our interpretation is reasonable
- Places where you think we could add sample applications, or expand upon the existing material
- Samples that do not work due to "shifting sands" of the underlying environment (e.g., changed APIs with new releases of an SDK)

However, those "softer" issues do not qualify for the formal bounty program.

Questions about the bug bounty, or problems you wish to report for bounty consideration, should be sent to bounty@commonsware.com.

Source Code License

The source code samples shown in this book are available for download from the [CommonsWare Web site](#) – just choose the tab of the book version you want, and click on the Source Code link for that tab. All of the Android projects are licensed under the [Apache 2.0 License](#), in case you have the desire to reuse any of it.

Creative Commons and the Four-to-Free (42F) Guarantee

Each CommonsWare book edition will be available for use under the [Creative Commons Attribution-Noncommercial-Share Alike 3.0](#) license as of the fourth anniversary of its publication date, or when 4,000 copies of the edition have been sold, whichever comes first. That means that, once four years have elapsed (perhaps sooner!), you can use this prose for non-commercial purposes. That is our Four-to-Free Guarantee to our readers and the broader community. For the purposes of this guarantee, new Warescriptions and renewals will be counted as sales of this edition, starting from the time the edition is published.

This edition of this book will be available under the aforementioned Creative Commons license on **December 1, 2012**. Of course, watch the CommonsWare Web site, as this edition might be relicensed sooner based on sales.

For more details on the Creative Commons Attribution-Noncommercial-Share Alike 3.0 license, visit the Creative Commons Web site.

Note that future editions of this book will become free on later dates, each four years from the publication of that edition or based on sales of that specific edition. Releasing one edition under the Creative Commons license does not automatically release *all* editions under that license.

Acknowledgments

I would like to thank the Android team, not only for putting out a good product, but for invaluable assistance on the Android Google Groups. In particular, I would like to thank Romain Guy, Justin@Google, and hackbod.

Icons used in the sample code were provided by the [Nuvola](#) icon set.

PART I – Core Concepts

CHAPTER 1

The Big Picture

Android devices, by and large, will be mobile phones. While the Android technology is being discussed for use in other areas (e.g., car dashboard "PCs"), for the most part, you can think of Android as being used on phones.

For developers, this has benefits and drawbacks.

On the plus side, circa 2008, Android-style smartphones are sexy. Offering Internet services over mobile devices dates back to the mid-1990's and the Handheld Device Markup Language (HDML). However, only in recent years have phones capable of Internet access taken off. Now, thanks to trends like text messaging and to products like Apple's iPhone, phones that can serve as Internet access devices are rapidly gaining popularity. So, working on Android applications gives you experience with an interesting technology (Android) in a fast-moving market segment (Internet-enabled phones), which is always a good thing.

The problem comes when you actually have to program the darn things.

Anyone with experience in programming for PDAs or phones has felt the pain of phones simply being *small* in all sorts of dimensions:

- Screens are small (you won't get comments like, "is that a 24-inch LCD in your pocket, or...?")
- Keyboards, if they exist, are small
- Pointing devices, if they exist, are annoying (as anyone who has lost their stylus will tell you) or inexact (large fingers and "multi-touch" LCDs are not a good mix)
- CPU speed and memory are tight compared to desktops and servers you may be used to
- You can have any programming language and development framework you want, so long as it was what the device manufacturer chose and burned into the phone's silicon
- And so on

Moreover, applications running on a phone have to deal with the fact that they're *on a phone*.

People with mobile phones tend to get very irritated when those phones don't work, which is why the "can you hear me now?" ad campaign from Verizon Wireless has been popular for the past few years. Similarly, those same people will get irritated at you if your program "breaks" their phone:

- ...by tying up the CPU such that calls can't be received
- ...by not working properly with the rest of the phone's OS, such that your application doesn't quietly fade to the background when a call comes in or needs to be placed
- ...by crashing the phone's operating system, such as by leaking memory like a sieve

Hence, developing programs for a phone is a different experience than developing desktop applications, Web sites, or back-end server processes. You wind up with different-looking tools, different-behaving frameworks, and "different than you're used to" limitations on what you can do with your program.

What Android tries to do is meet you halfway:

- You get a commonly-used programming language (Java) with some commonly used libraries (e.g., some Apache Commons APIs), with support for tools you may be used to (Eclipse)
- You get a fairly rigid and uncommon framework in which your programs need to run so they can be "good citizens" on the phone and not interfere with other programs or the operation of the phone itself

As you might expect, much of this book deals with that framework and how you write programs that work within its confines and take advantage of its capabilities.

What Androids Are Made Of

When you write a desktop application, you are "master of your own domain". You launch your main window and any child windows – like dialog boxes – that are needed. From your standpoint, you are your own world, leveraging features supported by the operating system, but largely ignorant of any other program that may be running on the computer at the same time. If you do interact with other programs, it is typically through an API, such as using JDBC (or frameworks atop it) to communicate with MySQL or another database.

Android has similar concepts, but packaged differently, and structured to make phones more crash-resistant.

Activities

The building block of the user interface is the **activity**. You can think of an activity as being the Android analogue for the window or dialog in a desktop application.

While it is possible for activities to not have a user interface, most likely your "headless" code will be packaged in the form of content providers or services, described below.

Content Providers

Content providers provide a level of abstraction for any data stored on the device that is accessible by multiple applications. The Android development model encourages you to make your own data available to other applications, as well as your own – building a content provider lets you do that, while maintaining complete control over how your data gets accessed.

Intents

Intents are system messages, running around the inside of the device, notifying applications of various events, from hardware state changes (e.g., an SD card was inserted), to incoming data (e.g., an SMS message arrived), to application events (e.g., your activity was launched from the device's main menu). Not only can you respond to intents, but you can create your own, to launch other activities, or to let you know when specific situations arise (e.g., raise such-and-so intent when the user gets within 100 meters of this-and-such location).

Services

Activities, content providers, and intent receivers are all short-lived and can be shut down at any time. Services, on the other hand, are designed to keep running, if needed, independent of any activity. You might use a service for checking for updates to an RSS feed, or to play back music even if the controlling activity is no longer operating.

Stuff At Your Disposal

Storage

You can package data files with your application, for things that do not change, such as icons or help files. You also can carve out a small bit of space on the device itself, for databases or files containing user-entered or

retrieved data needed by your application. And, if the user supplies bulk storage, like an SD card, you can read and write files on there as needed.

Network

Android devices will generally be Internet-ready, through one communications medium or another. You can take advantage of the Internet access at any level you wish, from raw Java sockets all the way up to a built-in WebKit-based Web browser widget you can embed in your application.

Multimedia

Android devices have the ability to play back and record audio and video. While the specifics may vary from device to device, you can query the device to learn its capabilities and then take advantage of the multimedia capabilities as you see fit, whether that is to play back music, take pictures with the camera, or use the microphone for audio note-taking.

GPS

Android devices will frequently have access to location providers, such as GPS, that can tell your applications where the device is on the face of the Earth. In turn, you can display maps or otherwise take advantage of the location data, such as tracking a device's movements if the device has been stolen.

Phone Services

And, of course, Android devices are typically phones, allowing your software to initiate calls, send and receive SMS messages, and everything else you expect from a modern bit of telephony technology.

Project Structure

The Android build system is organized around a specific directory tree structure for your Android project, much like any other Java project. The specifics, though, are fairly unique to Android and what it all does to prepare the actual application that will run on the device or emulator. Here's a quick primer on the project structure, to help you make sense of it all, particularly for the sample code referenced in this book.

Root Contents

...

The Sweat Off Your Brow

...

And Now, The Rest of the Story

...

What You Get Out Of It

...

Inside the Manifest

The foundation for any Android application is the manifest file: `AndroidManifest.xml` in the root of your project. Here is where you declare what all is inside your application – the activities, the services, and so on. You also indicate how these pieces attach themselves to the overall Android system; for example, you indicate which activity (or activities) should appear on the device's main menu (a.k.a., launcher).

When you create your application, you will get a starter manifest generated for you. For a simple application, offering a single activity and nothing else, the auto-generated manifest will probably work out fine, or perhaps require a few minor modifications. On the other end of the spectrum, the manifest file for the Android API demo suite is over 1,000 lines long. Your production Android applications will probably fall somewhere in the middle.

Most of the interesting bits of the manifest will be described in greater detail in the chapters on their associated Android features. For example, the service element will be described in greater detail in the chapter on creating services. For now, we just need to understand what the role of the manifest is and its general overall construction.

In The Beginning, There Was the Root, And It Was Good

...

Permissions, Instrumentations, and Applications (Oh, My!)

...

Your Application Does Something, Right?

...

Achieving the Minimum

...

Version=Control

...

Getting Going

As noted in the [preface](#), this book assumes you have downloaded the SDK (and perhaps the ADT plugin for Eclipse) and have it basically working in your environment. However, let's take a moment to discuss the notion of "targets" in Android, since they can be a bit confusing yet are rather important for your long-term application development.

Virtually There

...

Aiming at a Target

...

PART II – Activities

Creating a Skeleton Application

Every programming language or environment book starts off with the ever-popular "Hello, World!" demonstration: just enough of a program to prove you can build things, not so much that you cannot understand what is going on. However, the typical "Hello, World!" program has no interactivity (e.g., just dumps the words to a console), and so is really boring.

This chapter demonstrates a simple project, but one using Advanced Push-Button Technology™ and the current time, to show you how a simple Android activity works.

Begin at the Beginning

To work with anything in Android, you need a project. With ordinary Java, if you wanted, you could just write a program as a single file, compile it with `javac`, and run it with `java`, without any other support structures. Android is more complex, but to help keep it manageable, Google has supplied tools to help create the project. If you are using an Android-enabled IDE, such as Eclipse with the Android plugin, you can create a project inside of the IDE (e.g., select **File** > **New** > **Project**, then choose **Android** > **Android Project**).

If you are using tools that are not Android-enabled, you can use the `android create project` script, found in the `tools/` directory in your SDK installation. You will need to pass to `android create project` the API target

(see the [previous chapter](#)), the directory where you want the skeleton generated, the name of the default activity, and the Java package where all of this should reside:

```
android create project --target 2 \  
  --path /path/to/my/project/dir --activity Now \  
  --package com.commonsware.android.Now
```

You will wind up with a handful of pre-generated files, as described in a [previous chapter](#).

For the purposes of the samples shown in this book, you can download their project directories in a ZIP file on the CommonsWare Web site. These projects are ready for use; you do not need to run `android create project` on those unpacked samples.

The Activity

Your project's `src/` directory contains the standard Java-style tree of directories based upon the Java package you chose when you created the project (e.g., `com.commonsware.android` results in `src/com/commonsware/android/`). Inside the innermost directory you should find a pre-generated source file named `Now.java`, which is where your first activity will go.

Open `Now.java` in your editor and paste in the following code:

```
package com.commonsware.android.skeleton;  
  
import android.app.Activity;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.Button;  
import java.util.Date;  
  
public class Now extends Activity implements View.OnClickListener {  
    Button btn;  
  
    @Override  
    public void onCreate(Bundle icle) {  
        super.onCreate(icle);  
    }  
}
```

```
    btn = new Button(this);
    btn.setOnClickListener(this);
    updateTime();
    setContentView(btn);
}

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

private void updateTime() {
    btn.setText(new Date().toString());
}
}
```

Or, if you download the source files off the [Web site](#), you can just use the Skeleton/Now project directly.

Dissecting the Activity

Let's examine this piece by piece:

```
package com.commonware.android.skeleton;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import java.util.Date;
```

The package declaration needs to be the same as the one you used when creating the project. And, like any other Java project, you need to import any classes you reference. Most of the Android-specific classes are in the android package.

Remember that not every Java SE class is available to Android programs! Visit the [Android class reference](#) to see what is and is not available.

```
public class Now extends Activity implements View.OnClickListener {
    Button btn;
```

Activities are public classes, inheriting from the `android.Activity` base class. In this case, the activity holds a button (`btn`). Since, for simplicity, we want to trap all button clicks just within the activity itself, we also have the activity class implement `OnClickListener`.

```
@Override
public void onCreate(Bundle icle) {
    super.onCreate(icle);

    btn = new Button(this);
    btn.setOnClickListener(this);
    updateTime();
    setContentView(btn);
}
```

The `onCreate()` method is invoked when the activity is started. The first thing you should do is chain upward to the superclass, so the stock Android activity initialization can be done.

In our implementation, we then create the button instance (`new Button(this)`), tell it to send all button clicks to the activity instance itself (via `setOnClickListener()`), call a private `updateTime()` method (see below), and then set the activity's content view to be the button itself (via `setContentView()`).

We will discuss that magical `Bundle icle` in a later chapter. For the moment, consider it an opaque handle that all activities receive upon creation.

```
public void onClick(View view) {
    updateTime();
}
```

In Swing, a `JButton` click raises an `ActionEvent`, which is passed to the `ActionListener` configured for the button. In Android, a button click causes `onClick()` to be invoked in the `OnClickListener` instance configured for the button. The listener is provided the view that triggered the click (in this case, the button). All we do here is call that private `updateTime()` method:

```
private void updateTime() {  
    btn.setText(new Date().toString());  
}
```

When we open the activity (`onCreate()`) or when the button is clicked (`onClick()`), we update the button's label to be the current time via `setText()`, which functions much the same as the `JButton` equivalent.

Building and Running the Activity

To build the activity, either use your IDE's built-in Android packaging tool, or run `ant` in the base directory of your project. Then, to run the activity:

- Launch the emulator (e.g., run `tools/emulator` from your Android SDK installation)



Figure 1. The Android home screen

- Install the package (e.g., run `tools/adb install /path/to/this/example/bin/Now.apk` from your Android SDK installation)
- View the list of installed applications in the emulator and find the "Now" application

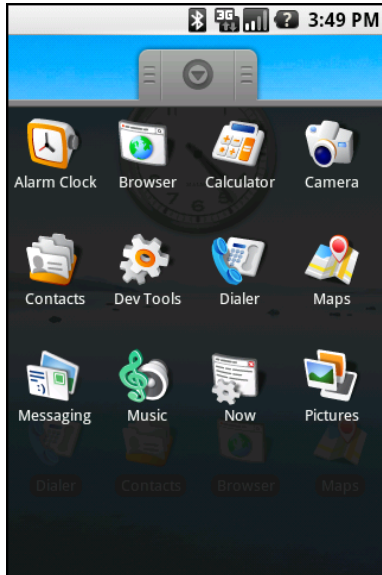


Figure 2. The Android application "launcher"

- Open that application

You should see an activity screen akin to:



Figure 3. The Now demonstration activity

Clicking the button – in other words, pretty much anywhere on the phone's screen – will update the time shown in the button's label.

Note that the label is centered horizontally and vertically, as those are the default styles applied to button captions. We can control that formatting, which will be covered in a [later chapter](#).

After you are done gazing at the awesomeness of Advanced Push-Button Technology™, you can click the back button on the emulator to return to the launcher.

Using XML-Based Layouts

While it is technically possible to create and attach widgets to our activity purely through Java code, the way we did in the [preceding chapter](#), the more common approach is to use an XML-based layout file. Dynamic instantiation of widgets is reserved for more complicated scenarios, where the widgets are not known at compile-time (e.g., populating a column of radio buttons based on data retrieved off the Internet).

With that in mind, it's time to break out the XML and learn how to lay out Android activity views that way.

What Is an XML-Based Layout?

...

Why Use XML-Based Layouts?

...

OK, So What Does It Look Like?

...

What's With the @ Signs?

...

And We Attach These to the Java...How?

...

The Rest of the Story

...

Employing Basic Widgets

Every GUI toolkit has some basic widgets: fields, labels, buttons, etc. Android's toolkit is no different in scope, and the basic widgets will provide a good introduction as to how widgets work in Android activities.

Assigning Labels

...

Button, Button, Who's Got the Button?

...

Fleeting Images

...

Fields of Green. Or Other Colors.

...

Just Another Box to Check

...

Turn the Radio Up

...

It's Quite a View

...

Useful Properties

...

Useful Methods

...

Colors

...

Working with Containers

Containers pour a collection of widgets (and possibly child containers) into specific layouts you like. If you want a form with labels on the left and fields on the right, you will need a container. If you want OK and Cancel buttons to be beneath the rest of the form, next to one another, and flush to right side of the screen, you will need a container. Just from a pure XML perspective, if you have multiple widgets (beyond `RadioButton` widgets in a `RadioGroup`), you will need a container just to have a root element to place the widgets inside.

Most GUI toolkits have some notion of layout management, frequently organized into containers. In Java/Swing, for example, you have layout managers like `BoxLayout` and containers that use them (e.g., `Box`). Some toolkits stick strictly to the box model, such as XUL and Flex, figuring that any desired layout can be achieved through the right combination of nested boxes.

Android, through `LinearLayout`, also offers a "box" model, but in addition supports a range of containers providing different layout rules. In this chapter, we will look at three commonly-used containers: `LinearLayout` (the box model), `RelativeLayout` (a rule-based model), and `TableLayout` (the grid model), along with `ScrollView`, a container designed to assist with implementing scrolling containers. In the [next chapter](#), we will examine some more esoteric containers.

Thinking Linearly

...

Concepts and Properties

...

Orientation

...

Fill Model

...

Weight

...

Gravity

...

Padding

...

Example

...

All Things Are Relative

...

Concepts and Properties

...

Positions Relative to Container

...

Relative Notation in Properties

...

Positions Relative to Other Widgets

...

Order of Evaluation

...

Example

...

Tabula Rasa

...

Concepts and Properties

...

Putting Cells in Rows

...

Non-Row Children of TableLayout

...

Stretch, Shrink, and Collapse

...

Example

...

Scrollwork

...

Using Selection Widgets

Back in the chapter on [basic widgets](#), you saw how fields could have constraints placed upon them to limit possible input, such as numeric-only or phone-number-only. These sorts of constraints help users "get it right" when entering information, particularly on a mobile device with cramped keyboards.

Of course, the ultimate in constrained input is to select a choice from a set of items, such as the radio buttons seen earlier. Classic UI toolkits have listboxes, comboboxes, drop-down lists, and the like for that very purpose. Android has many of the same sorts of widgets, plus others of particular interest for mobile devices (e.g., the `Gallery` for examining saved photos).

Moreover, Android offers a flexible framework for determining what choices are available in these widgets. Specifically, Android offers a framework of data adapters that provide a common interface to selection lists ranging from static arrays to database contents. Selection views – widgets for presenting lists of choices – are handed an adapter to supply the actual choices.

Adapting to the Circumstances

...

Using ArrayAdapter

...

Other Key Adapters

...

Lists of Naughty and Nice

...

Selection Modes

...

Spin Control

...

Grid Your Lions (Or Something Like That...)

...

Fields: Now With 35% Less Typing!

...

Galleries, Give Or Take The Art

...

Getting Fancy With Lists

The humble `ListView` is one of the most important widgets in all of Android, simply because it is used so frequently. Whether choosing a contact to call or an email message to forward or an ebook to read, `ListView` widgets are employed in a wide range of activities.

Of course, it would be nice if they were more than just plain text.

The good news is that they can be as fancy as you want, within the limitations of a mobile device's screen, of course. However, making them fancy takes some work and some features of Android that we will cover in this chapter.

Getting To First Base

...

A Dynamic Presentation

...

A Sidebar About Inflation

...

And Now, Back To Our Story

...

Better. Stronger. Faster.

...

Using convertView

...

Using the Holder Pattern

...

Making a List...

...

...And Checking It Twice

...

Adapting Other Adapters

...

Employing Fancy Widgets and Containers

The widgets and containers covered to date are not only found in many GUI toolkits (in one form or fashion), but also are widely used in building GUI applications, whether Web-based, desktop, or mobile. The widgets and containers in this chapter are a little less widely used, though you will likely find many to be quite useful.

Pick and Choose

...

Time Keeps Flowing Like a River

...

Making Progress

...

Putting It On My Tab

...

The Pieces

...

The Idiosyncrasies

...

Wiring It Together

...

Adding Them Up

...

Intents and Views

...

Flipping Them Off

...

Getting In Somebody's Drawer

...

Other Good Stuff

...

The Input Method Framework

Android 1.5 introduced the input method framework (IMF), which is commonly referred to as "soft keyboards". However, the "soft keyboard" term is not necessarily accurate, as IMF could be used for handwriting recognition or other means of accepting text input via the screen.

Keyboards, Hard and Soft

...

Tailored To Your Needs

...

Tell Android Where It Can Go

...

Fitting In

...

Unleash Your Inner Dvorak

...

Applying Menus

Like applications for the desktop and some mobile operating systems, such as PalmOS and Windows Mobile, Android supports activities with "application" menus. Some Android phones will have a dedicated menu key for popping up the menu; others will offer alternate means for triggering the menu to appear.

Also, as with many GUI toolkits, you can create "context menus". On a traditional GUI, this might be triggered by the right-mouse button. On mobile devices, context menus typically appear when the user "taps-and-holds" over a particular widget. For example, if a `TextView` had a context menu, and the device was designed for finger-based touch input, you could push the `TextView` with your finger, hold it for a second or two, and a pop-up menu will appear for the user to choose from.

Where Android differs from most other GUI toolkits is in terms of menu construction. While you can add items to the menu, you do not have full control over the menu's contents, nor the timing of when the menu is built. Part of the menu is system-defined, and that portion is managed by the Android framework itself.

Flavors of Menu

...

Menus of Options

...

Menus in Context

...

Taking a Peek

...

Yet More Inflation

...

Menu XML Structure

...

Menu Options and XML

...

Title

...

Icon

...

Order

...

Enabled

...

Visible

...

Shortcut

...

Inflating the Menu

...

Inevitably, you'll get the question "hey, can we change this font?" when doing application development. The answer depends on what fonts come with the platform, whether you can add other fonts, and how to apply them to the widget or whatever needs the font change.

Android is no different. It comes with some fonts plus a means for adding new fonts. Though, as with any new environment, there are a few idiosyncrasies to deal with.

Love The One You're With

...

Here a Glyph, There a Glyph

...

Embedding the WebKit Browser

Other GUI toolkits let you use HTML for presenting information, from limited HTML renderers (e.g., Java/Swing, wxWidgets) to embedding Internet Explorer into .NET applications. Android is much the same, in that you can embed the built-in Web browser as a widget in your own activities, for displaying HTML or full-fledged browsing. The Android browser is based on WebKit, the same engine that powers Apple's Safari Web browser.

The Android browser is sufficiently complex that it gets its own Java package (`android.webkit`), though using the `webView` widget itself can be simple or powerful, based upon your requirements.

A Browser, Writ Small

...

Loading It Up

...

Navigating the Waters

...

Entertaining the Client

...

Settings, Preferences, and Options (Oh, My!)

...

Showing Pop-Up Messages

Sometimes, your activity (or other piece of Android code) will need to speak up.

Not every interaction with Android users will be neat, tidy, and containable in activities composed of views. Errors will crop up. Background tasks may take way longer than expected. Something asynchronous may occur, such as an incoming message. In these and other cases, you may need to communicate with the user outside the bounds of the traditional user interface.

Of course, this is nothing new. Error messages in the form of dialog boxes have been around for a very long time. More subtle indicators also exist, from task tray icons to bouncing dock icons to a vibrating cell phone.

Android has quite a few systems for letting you alert your users outside the bounds of an Activity-based UI. One, notifications, is tied heavily into intents and services and, as such, is covered in a [later chapter](#). In this chapter, you will see two means of raising pop-up messages: toasts and alerts.

Raising Toasts

...

Alert! Alert!

...

Checking Them Out

...

Dealing with Threads

Ideally, you want your activities to be downright snappy, so your users don't feel that your application is sluggish. Responding to user input quickly (e.g., zooms) is a fine goal. At minimum, though, you need to make sure you respond within 5 seconds, lest the `ActivityManager` decide to play the role of the Grim Reaper and kill off your activity as being non-responsive.

Of course, your activity might have real work to do, which takes non-negligible amounts of time. There are two ways of dealing with this:

1. Do expensive operations in a background service, relying on **notifications** to prompt users to go back to your activity
2. Do expensive work in a background thread

Android provides a veritable cornucopia of means to set up background threads yet allow them to safely interact with the UI on the UI thread. These include `Handler` objects and posting `Runnable` objects to the `View`.

Getting Through the Handlers

...

Messages

...

Runnables

...

Running In Place

...

Where, Oh Where Has My UI Thread Gone?

...

Asynching Situation

...

The Theory

...

AsyncTask, Generics, and Varargs

...

The Stages of AsyncTask

...

A Sample Task

...

The AddStringTask Declaration

...

The doInBackground() Method

...

The onProgressUpdate() Method

...

The onPostExecute() Method

...

The Activity

...

The Results

...

And Now, The Caveats

...

Handling Activity Lifecycle Events

While this may sound like a broken record...please remember that Android devices, by and large, are phones. As such, some activities are more important than others – taking a call is probably more important to users than is playing Sudoku. And, since it is a phone, it probably has less RAM than does your current desktop or notebook.

As a result, your activity may find itself being killed off because other activities are going on and the system needs your activity's memory. Think of it as the Android equivalent of the "circle of life" – your activity dies so others may live, and so on. You cannot assume that your activity will run until you think it is complete, or even until the user thinks it is complete.

This is one example – perhaps the most important example – of how an activity's lifecycle will affect your own application logic. This chapter covers the various states and callbacks that make up an activity's lifecycle and how you can hook into them appropriately.

Schroedinger's Activity

...

Life, Death, and Your Activity

...

onCreate() and onDestroy()

...

onStart(), onRestart(), and onStop()

...

onPause() and onResume()

...

The Grace of State

...

PART III – Data Stores, Network Services, and APIs

Using Preferences

Android has many different ways for you to store data for long-term use by your activity. The simplest to use is the preferences system.

Android allows activities and applications to keep preferences, in the form of key/value pairs (akin to a `Map`), that will hang around between invocations of an activity. As the name suggests, the primary purpose is for you to store user-specified configuration details, such as the last feed the user looked at in your feed reader, or what sort order to use by default on a list, or whatever. Of course, you can store in the preferences whatever you like, so long as it is keyed by a `String` and has a primitive value (`boolean`, `String`, etc.)

Preferences can either be for a single activity or shared among all activities in an application. Eventually, preferences might be shareable across applications, but that is not supported as of the time of this writing.

Getting What You Want

...

Stating Your Preference

...

And Now, a Word From Our Framework

...

Letting Users Have Their Say

...

Adding a Wee Bit O' Structure

...

The Kind Of Pop-Ups You Like

...

Accessing Files

While Android offers structured storage, via [preferences](#) and [databases](#), sometimes a simple file will suffice. Android offers two models for accessing files: one for files pre-packaged with your application, and one for files created on-device by your application.

You And The Horse You Rode In On

...

Readin' 'n Writin'

...

Working with Resources

Resources are static bits of information held outside the Java source code. You have seen one type of resource – the layout – frequently in the examples in this book. There are many other types of resource, such as images and strings, that you can take advantage of in your Android applications.

The Resource Lineup

...

String Theory

...

Plain Strings

...

String Formats

...

Styled Text

...

Styled Formats

...

Got the Picture?

...

XML: The Resource Way

...

Miscellaneous Values

...

Dimensions

...

Colors

...

Arrays

...

Different Strokes for Different Folks

...

Managing and Accessing Local Databases

SQLite is a very popular embedded database, as it combines a clean SQL interface with a very small memory footprint and decent speed. Moreover, it is public domain, so everyone can use it. Lots of firms (Adobe, Apple, Google, Sun, Symbian) and open source projects (Mozilla, PHP, Python) all ship products with SQLite.

For Android, SQLite is "baked into" the Android runtime, so every Android application can create SQLite databases. Since SQLite uses a SQL interface, it is fairly straightforward to use for people with experience in other SQL-based databases. However, its native API is not JDBC, and JDBC might be too much overhead for a memory-limited device like a phone, anyway. Hence, Android programmers have a different API to learn – the good news being is that it is not that difficult.

This chapter will cover the basics of SQLite use in the context of working on Android. It by no means is a thorough coverage of SQLite as a whole. If you want to learn more about SQLite and how to use it in other environment than Android, a fine book is [The Definitive Guide to SQLite](#) by Michael Owens.

Activities will typically access a database via a content provider or service. As such, this chapter does not have a full example. You will find a full

example of a content provider that accesses a database in the [Building a Content Provider](#) chapter.

A Quick SQLite Primer

...

Start at the Beginning

...

Setting the Table

...

Makin' Data

...

What Goes Around, Comes Around

...

Raw Queries

...

Regular Queries

...

Building with Builders

...

Using Cursors

...

Making Your Own Cursors

...

Data, Data, Everywhere

...

Leveraging Java Libraries

Java has as many, if not more, third-party libraries than any other modern programming language. Here, "third-party libraries" refer to the innumerable JARs that you can include in a server or desktop Java application – the things that the Java SDKs themselves do not provide.

In the case of Android, the Dalvik VM at its heart is not precisely Java, and what it provides in its SDK is not precisely the same as any traditional Java SDK. That being said, many Java third-party libraries still provide capabilities that Android lacks natively and therefore may be of use to you in your project, for the ones you can get working with Android's flavor of Java.

This chapter explains what it will take for you to leverage such libraries and the limitations on Android's support for arbitrary third-party code.

The Outer Limits

...

Ants and Jars

...

Following the Script

...

...And Not A Drop To Drink

...

Reviewing the Script

...

Communicating via the Internet

The expectation is that most, if not all, Android devices will have built-in Internet access. That could be WiFi, cellular data services (EDGE, 3G, etc.), or possibly something else entirely. Regardless, most people – or at least those with a data plan or WiFi access – will be able to get to the Internet from their Android phone.

Not surprisingly, the Android platform gives developers a wide range of ways to make use of this Internet access. Some offer high-level access, such as the integrated WebKit browser component we saw in an [earlier chapter](#). If you want, you can drop all the way down to using raw sockets. Or, in between, you can leverage APIs – both on-device and from 3rd-party JARs – that give you access to specific protocols: HTTP, XMPP, SMTP, and so on.

The emphasis of this book is on the higher-level forms of access: the WebKit component and Internet-access APIs, as busy coders should be trying to reuse existing components versus rolling one's own on-the-wire protocol wherever possible.

REST and Relaxation

...

HTTP Operations via Apache HttpComponents

...

Parsing Responses

...

Stuff To Consider

...

PART IV – Intents

Creating Intent Filters

Up to now, the focus of this book has been on activities opened directly by the user from the device's launcher. This, of course, is the most obvious case for getting your activity up and visible to the user. And, in many cases it is the primary way the user will start using your application.

However, remember that the Android system is based upon lots of loosely-coupled components. What you might accomplish in a desktop GUI via dialog boxes, child windows, and the like are mostly supposed to be independent activities. While one activity will be "special", in that it shows up in the launcher, the other activities all need to be reached...somehow.

The "how" is via intents.

An intent is basically a message that you pass to Android saying, "Yo! I want to do...er...something! Yeah!" How specific the "something" is depends on the situation – sometimes you know exactly what you want to do (e.g., open up one of your other activities), and sometimes you don't.

In the abstract, Android is all about intents and receivers of those intents. So, now that we are well-versed in creating activities, let's dive into intents, so we can create more complex applications while simultaneously being "good Android citizens".

What's Your Intent?

...

Pieces of Intents

...

Intent Routing

...

Stating Your Intent(ions)

...

Narrow Receivers

...

The Pause Caveat

...

Hence, you can only really use the Intent framework as an arbitrary message bus if:

- Your receiver does not care if it misses messages because it was not active, or
- You provide some means of getting the receiver "caught up" on messages it missed while it was inactive

In the chapters on **creating** and **using** services, you will see an example of the former condition, where the receiver (service client) will use Intent-based messages when they are available but does not need them if the client is not active.

Launching Activities and Sub-Activities

As discussed previously, the theory behind the Android UI architecture is that developers should decompose their application into distinct activities, each implemented as an `Activity`, each reachable via intents, with one "main" activity being the one launched by the Android launcher. For example, a calendar application could have activities for viewing the calendar, viewing a single event, editing an event (including adding a new one), and so forth.

This, of course, implies that one of your activities has the means to start up another activity. For example, if somebody clicks on an event from the view-calendar activity, you might want to show the view-event activity for that event. This means that, somehow, you need to be able to cause the view-event activity to launch and show a specific event (the one the user clicked upon).

This can be further broken down into two scenarios:

1. You know what activity you want to launch, probably because it is another activity in your own application
2. You have a content `Uri` to...something, and you want your users to be able to do...something with it, but you do not know up front what the options are

This chapter covers the first scenario; the next chapter handles the second.

Peers and Subs

...

Start 'Em Up

...

Make an Intent

...

Make the Call

...

Tabbed Browsing, Sort Of

...

Handling Rotation

Some Android handsets, like the T-Mobile G1, offer a slide-out keyboard that triggers rotating the screen from portrait to landscape. Other handsets might use accelerometers to determine screen rotation, like the iPhone does. As a result, it is reasonable to assume that switching from portrait to landscape and back again may be something your users will look to do.

Android has a number of ways for you to handle screen rotation, so your application can properly handle either orientation. All these facilities do is help you detect and manage the rotation process – you are still required to make sure you have layouts that look decent on each orientation.

A Philosophy of Destruction

...

It's All The Same, Just Different

...

Now With More Savings!

...

DIY Rotation

...

Forcing the Issue

...

Making Sense of it All

...

PART V – Content Providers and Services

Using a Content Provider

Any `Uri` in Android that begins with the `content://` scheme represents a resource served up by a content provider. Content providers offer data encapsulation using `Uri` instances as handles – you neither know nor care where the data represented by the `Uri` comes from, so long as it is available to you when needed. The data could be stored in a SQLite database, or in flat files, or retrieved off a device, or be stored on some far-off server accessed over the Internet.

Given a `Uri`, you can perform basic CRUD (create, read, update, delete) operations using a content provider. `Uri` instances can represent either collections or individual pieces of content. Given a collection `Uri`, you can create new pieces of content via insert operations. Given an instance `Uri`, you can read data represented by the `Uri`, update that data, or delete the instance outright.

Android lets you use existing content providers, plus create your own. This chapter covers using content providers; the [next chapter](#) will explain how you can serve up your own data using the content provider framework.

Pieces of Me

...

Getting a Handle

...

Makin' Queries

...

Adapting to the Circumstances

...

Doing It By Hand

...

Position

...

Getting Properties

...

Give and Take

...

Beware of the BLOB!

...

Building a Content Provider

Building a content provider is probably the most complicated and tedious task in all of Android development. There are many requirements of a content provider, in terms of methods to implement and public data members to supply. And, until you try using it, you have no great way of telling if you did any of it correctly (versus, say, building an activity and getting validation errors from the resource compiler).

That being said, building a content provider is of huge importance if your application wishes to make data available to other applications. If your application is keeping its data solely to itself, you may be able to avoid creating a content provider, just accessing the data directly from your activities. But, if you want your data to possibly be used by others – for example, you are building a feed reader and you want other programs to be able to access the feeds you are downloading and caching – then a content provider is right for you.

First, Some Dissection

...

Next, Some Typing

...

Step #1: Create a Provider Class

...

onCreate()

...

query()

...

insert()

...

update()

...

delete()

...

getType()

...

Step #2: Supply a Uri

...

Step #3: Declare the Properties

...

Step #4: Update the Manifest

...

Notify-On-Change Support

...

Requesting and Requiring Permissions

In the late 1990's, a wave of viruses spread through the Internet, delivered via email, using contact information culled from Microsoft Outlook. A virus would simply email copies of itself to each of the Outlook contacts that had an email address. This was possible because, at the time, Outlook did not take any steps to protect data from programs using the Outlook API, since that API was designed for ordinary developers, not virus authors.

Nowadays, many applications that hold onto contact data secure that data by requiring that a user explicitly grant rights for other programs to access the contact information. Those rights could be granted on a case-by-case basis or a once at install time.

Android is no different, in that it requires permissions for applications to read or write contact data. Android's permission system is useful well beyond contact data, and for content providers and services beyond those supplied by the Android framework.

You, as an Android developer, will frequently need to ensure your applications have the appropriate permissions to do what you want to do with other applications' data. You may also elect to require permissions for other applications to use your data or services, if you make those available to other Android components. This chapter covers how to accomplish both these ends.

Mother, May I?

...

Halt! Who Goes There?

...

Enforcing Permissions via the Manifest

...

Enforcing Permissions Elsewhere

...

May I See Your Documents?

...

Creating a Service

As noted previously, Android services are for long-running processes that may need to keep running even when decoupled from any activity. Examples include playing music even if the "player" activity gets garbage-collected, polling the Internet for RSS/Atom feed updates, and maintaining an online chat connection even if the chat client loses focus due to an incoming phone call.

Services are created when manually started (via an API call) or when some activity tries connecting to the service via inter-process communication (IPC). Services will live until no longer needed and if RAM needs to be reclaimed, or until shut down (on their own volition or because nobody is using them anymore). Running for a long time isn't without its costs, though, so services need to be careful not to use too much CPU or keep radios active too much of the time, lest the service cause the device's battery to get used up too quickly.

This chapter covers how you can create your own services; the [next chapter](#) covers how you can use such services from your activities or other contexts. Both chapters will analyze the `Service/WeatherPlus` sample application, with this chapter focusing mostly on the `WeatherPlusService` implementation. `WeatherPlusService` extends the weather-fetching logic of the original `Internet/Weather` sample, by bundling it in a service that monitors changes in location, so the weather is updated as the emulator is "moved".

Service with Class

...

There Can Only Be One

...

Manifest Destiny

...

Lobbing One Over the Fence

...

Callbacks

...

Broadcast Intents

...

Where's the Remote? And the Rest of the Code?

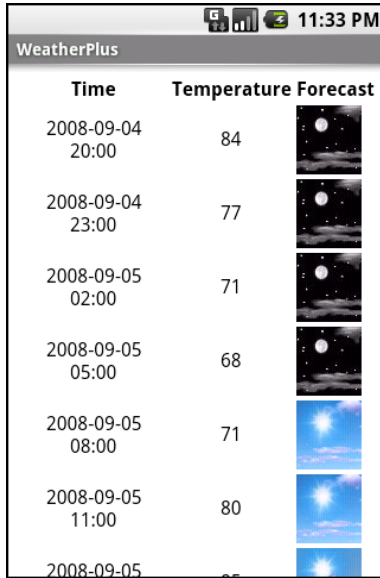
...

Invoking a Service

Services can be used by any application component that "hangs around" for a reasonable period of time. This includes activities, content providers, and other services. Notably, it does not include pure broadcast receivers (i.e., intent receivers that are not part of an activity), since those will get garbage collected immediately after each instance processes one incoming Intent.

To use a local service, you need to start the service, get access to the service object, then call methods on that service. You can then stop the service when you are done with it, or perhaps let the service stop itself. Using remote services is somewhat more complex, which is why a discussion of it is reserved for *The Busy Coder's Guide to Advanced Android Development* companion volume.

In this chapter, we will look at the client side of the `Service/WeatherPlus` sample application. The `WeatherPlus` activity looks an awful lot like the original weather application – just a Web page showing a weather forecast:



The screenshot shows the WeatherPlus application interface. At the top, there is a status bar with icons for signal strength, battery, and time (11:33 PM). Below the status bar, the application title "WeatherPlus" is displayed. The main content area contains a table with the following data:

Time	Temperature	Forecast
2008-09-04 20:00	84	[Night sky with moon]
2008-09-04 23:00	77	[Night sky with moon]
2008-09-05 02:00	71	[Night sky with moon]
2008-09-05 05:00	68	[Night sky with moon]
2008-09-05 08:00	71	[Day sky with sun]
2008-09-05 11:00	80	[Day sky with sun]
2008-09-05		[Day sky with sun]

Figure 4. The WeatherPlus service client

The difference is that, as the emulator "moves", the weather forecast changes, based on updates provided by the service.

Manual Transmission

...

Catching the Lob

...

Alerting Users Via Notifications

Pop-up messages. Tray icons and their associated "bubble" messages. Bouncing dock icons. You are no doubt used to programs trying to get your attention, sometimes for good reason.

Your phone also probably chirps at you for more than just incoming calls: low battery, alarm clocks, appointment notifications, incoming text message or email, etc.

Not surprisingly, Android has a whole framework for dealing with these sorts of things, collectively called "notifications".

Types of Pestering

...

Hardware Notifications

...

Icons

...

Seeing Pestering in Action

...

PART VI – Other Android Capabilities and Tools

Accessing Location-Based Services

A popular feature on current-era mobile devices is GPS capability, so the device can tell you where you are at any point in time. While the most popular use of GPS service is mapping and directions, there are other things you can do if you know your location. For example, you might set up a dynamic chat application where the people you can chat with are based on physical location, so you're chatting with those you are nearest. Or, you could automatically "geotag" posts to Twitter or similar services.

GPS is not the only way a mobile device can identify your location. Alternatives include:

- The European equivalent to GPS, called Galileo, which is still under development at the time of this writing
- Cell tower triangulation, where your position is determined based on signal strength to nearby cell towers
- Proximity to public WiFi "hotspots" that have known geographic locations

Android devices may have one or more of these services available to them. You, as a developer, can ask the device for your location, plus details on what providers are available. There are even ways for you to simulate your location in the emulator, for use in testing your location-enabled applications.

Location Providers: They Know Where You're Hiding

...

Finding Yourself

...

On the Move

...

Are We There Yet? Are We There Yet? Are We There Yet?

...

Testing...Testing...

...

Mapping with MapView and MapActivity

One of Google's most popular services – after search, of course – is Google Maps, where you can find everything from the nearest pizza parlor to directions from New York City to San Francisco (only 2,905 miles!) to street views and satellite imagery.

Android, not surprisingly, integrates Google Maps. There is a mapping activity available to users straight off the main Android launcher. More relevant to you, as a developer, are `MapView` and `MapActivity`, which allow you to integrate maps into your own applications. Not only can you display maps, control the zoom level, and allow people to pan around, but you can tie in Android's [location-based services](#) to show where the device is and where it is going.

Fortunately, integrating basic mapping features into your Android project is fairly easy. However, there is a fair bit of power available to you, if you want to get fancy.

Terms, Not of Endearment

...

Piling On

...

The Bare Bones

...

Exercising Your Control

...

Zoom

...

Center

...

Rugged Terrain

...

Layers Upon Layers

...

Overlay Classes

...

Drawing the ItemizedOverlay

...

Handling Screen Taps

...

My, Myself, and MyLocationOverlay

...

The Key To It All

...

Handling Telephone Calls

Many, if not most, Android devices will be phones. As such, not only will users be expecting to place and receive calls using Android, but you will have the opportunity to help them place calls, if you wish.

Why might you want to?

- Maybe you are writing an Android interface to a sales management application (a la Salesforce.com) and you want to offer users the ability to call prospects with a single button click, and without them having to keep those contacts both in your application and in the phone's contacts application
- Maybe you are writing a social networking application, and the roster of phone numbers that you can access shifts constantly, so rather than try to "sync" the social network contacts with the phone's contact database, you let people place calls directly from your application
- Maybe you are creating an alternative interface to the existing contacts system, perhaps for users with reduced motor control (e.g., the elderly), sporting big buttons and the like to make it easier for them to place calls

Whatever the reason, Android has the means to let you manipulate the phone just like any other piece of the Android system.

Report To The Manager

...

You Make the Call!

...

Searching with SearchManager

One of the firms behind the Open Handset Alliance – Google – has a teeny weeny Web search service, one you might have heard of in passing. Given that, it's not surprising that Android has some amount of built-in search capabilities.

Specifically, Android has "baked in" the notion of searching not only on the device for data, but over the air to Internet sources of data.

Your applications can participate in the search process, by triggering searches or perhaps by allowing your application's data to be searched.

Note that this is fairly new to the Android platform, and so some shifting in the APIs is likely. Stay tuned for updates to this chapter.

Hunting Season

...

Search Yourself

...

Craft the Search Activity

...

Update the Manifest

...

Searching for Meaning In Randomness

...

Development Tools

The Android SDK is more than a library of Java classes and API calls. It also includes a number of tools to assist in application development.

Much of the focus has been on the Eclipse plug-in, to integrate Android development with that IDE. Secondary emphasis has been placed on the plug-in's equivalents for use in other IDEs or without an IDE, such as `adb` for communicating with a running emulator.

This chapter will cover other tools beyond those two groups.

Hierarchical Management

...

Delightful Dalvik Debugging Detailed, De-moed

...

Logging

...

File Push and Pull

...

Screenshots

...

Location Updates

...

Placing Calls and Messages

...

Put It On My Card

...

Creating a Card Image

...

"Inserting" the Card

...

Handling Platform Changes

Android is going to undergo rapid evolution over the next few years. Perhaps, in time, the rate of change will decline some. However, for the here and now, you have to assume significant Android releases every 6-12 months, and changes to the lineup of possible Android hardware on an ongoing basis. So, while right now, the focus of Android is phones, soon you will see Android netbooks, Android tablets, Android media players, and so on.

Many of these changes will have little impact on your existing code. Some, though, will necessitate at least new rounds of testing for your applications, and perhaps changes to those applications based upon the test results.

In this chapter, we cover a number of the areas which may cause you trouble in the future as Android evolves.

This Application Contains Explicit...Instructions

...

Brand Management

...

More Things That Make You Go "Boom"

...

View Hierarchy

...

Changing APIs and Resources

...

Where Do We Go From Here?

Obviously, this book does not cover everything. And while your #1 resource (besides the book) is going to be the Android SDK documentation, you are likely to need information beyond what's covered in either of those places.

Searching online for "android" and a class name is a good way to turn up tutorials that reference a given Android class. However, bear in mind that tutorials written before late August 2008 are probably written for the M5 SDK and, as such, will require considerable adjustment to work properly in current SDKs.

Beyond randomly hunting around for tutorials, though, this chapter outlines some other resources to keep in mind.

Questions. Sometimes, With Answers.

...

Heading to the Source

...

Getting Your News Fix

...

Keyword Index

Class	
AbsoluteLayout.....	150, 151
ActionEvent.....	26
ActionListener.....	26
Activity....	8, 80, 195, 196, 204, 206, 216, 223, 224, 243, 273, 309, 311, 318, 331
ActivityManager.....	201
Adapter.....	100, 102, 103
AdapterWrapper.....	119, 120
AddStringsTask.....	213
AddStringTask.....	210, 211
AlertDialog.....	196, 197
AnalogClock.....	130, 139
android.text.Spanned.....	252
AndroidBrowser.....	319
ArrayAdapter...78, 79, 81, 82, 90, 101, 112, 114, 115, 124, 210, 212, 242	
ArrayList.....	242
AssetManager.....	181
AsyncTask.....	206-212, 374, 375
AutoCompleteTextView.....	44, 92-94
BaseColumns.....	359
Box.....	53
BoxLayout.....	53
BroadcastReceiver.....	308, 309, 314, 380
BrowserTab.....	321
Builder.....	196, 197
Bundle.....	217, 219, 305, 314, 324, 328, 330
Button.....	31, 33-36, 40, 41, 141, 142, 145, 146, 148, 253, 257, 433, 434, 452, 453
Calendar.....	128
CheckAdapter.....	114
CheckBox.....	45, 48, 50
CheckBoxPreference.....	225
ColorStateList.....	51
CompoundButton.....	48
ConstantsBrowser.....	342, 343, 346
ContentManager.....	382
ContentObserver.....	360, 361
ContentProvider.....	278, 346, 347, 351
ContentResolver.....	346, 360, 361

Keyword Index

ContentValues.....	275, 346, 354, 355, 359	Forecast.....	297, 375
Context.....	78, 196, 223, 224, 243, 273, 343	FrameLayout.....	133-135, 141, 147
ContextMenu.....	166, 167	Gallery.....	77, 96
ContextMenu.ContextMenuInfo.....	166, 167	GeoPoint.....	403
Criteria.....	393	GridView.....	88, 89, 96
Criteria.....	393	Handler.....	201-206, 214, 218, 379
Cursor....	79, 105, 114, 124, 277, 279, 280, 341, 343-347, 353, 354, 359	HelpActivity.....	313
CursorAdapter.....	79, 124	HorizontalScrollView.....	75
CWBrowser.....	319	HttpClient.....	294, 296, 298, 299, 370
DatabaseHelper.....	352	HttpGet.....	294, 296
DateFormat.....	128	HttpPost.....	294
DatePicker.....	125	HttpRequest.....	294
DatePickerDialog.....	125, 128	HttpResponse.....	294
DefaultHttpClient.....	294	IBinder.....	371
DialogWrapper.....	346	ImageButton.....	41, 256, 257
DigitalClock.....	130	ImageView.....	41, 104, 108, 111, 148, 256, 347
Document.....	242	IMEDemo1.....	160
Double.....	313	IMEDemo2.....	160, 161
Drawable.....	97, 135, 256, 329, 383, 405, 406	InputMethod.....	43
DrawerDemo.....	150	InputStream.....	239, 242, 243, 297
EditPreferences.....	226	InputStreamReader.....	243
EditText. .42-44, 92, 93, 125, 153, 154, 156, 158-161, 325, 433		Integer.....	115
EditTextPreference.....	235	Intent....	132, 141, 190, 191, 305, 309, 313, 314, 318, 321, 351, 373-375, 377-380, 395, 396, 412, 423
Exception.....	182	Interpreter.....	286
ExpandableListView.....	151	ItemizedOverlay.....	405, 407
FancyLists/ViewWrapper.....	109	Iterator.....	345
FetchForecastTask.....	374	JButton.....	26, 27
FieldDemo.....	154	JCheckBox.....	78
FlowLayout.....	54	JComboBox.....	84
		JLabel.....	78

Keyword Index

JList.....	78	NooYawk.....	404, 408
JTabbedPane.....	132	Notification.....	373, 382, 383, 385
JTable.....	78	NotificationManager.....	382, 385
LatinIME.....	162	NotifyDemo.....	383
LayoutInflater.....	103, 104, 121, 139	NotifyMessage.....	385
LinearLayout.....	53-58, 70, 100, 104, 115, 121, 135, 147, 433	Now.....	35, 36
List.....	167, 393	NowRedux.....	35
ListActivity.....	80-82, 134, 209, 400	Object.....	329
ListAdapter.....	118-120, 123, 151, 422, 423	OnCheckedChangeListener.....	45, 46, 60
ListCellRenderer.....	78	OnClickListener.....	26, 128, 199, 315
ListDemo.....	170	OnDateChangeListener.....	126
ListPreference.....	235	OnDateSetListener.....	126, 128
ListView.....	79, 80, 82, 84, 85, 96, 99-101, 104, 106, 111, 112, 118, 123, 167, 295, 343, 344, 346, 400, 420, 422	OnItemSelectedListener.....	86
Location.....	295, 393-395	OnTimeChangeListener.....	126
LocationListener.....	394, 395	OnTimeSetListener.....	126, 128
LocationManager.....	370, 392-395	OutputStream.....	243
LocationProvider.....	371, 392-394, 441	OutputStreamWriter.....	243
LoremBase.....	421	Overlay.....	405, 407
LoremDemo.....	422, 425	OverlayItem.....	405, 407
LoremSearch.....	425	PendingIntent.....	383, 385, 395, 396
Map.....	223, 275	PhoneWindowsDecorView.....	452
MapActivity.....	399-402	Preference.....	225
MapController.....	402, 403	PreferenceActivity.....	232
MapView.....	399, 401, 402, 404, 405, 408, 409	PreferenceCategory.....	231, 232
Menu.....	164, 166, 175, 176	PreferenceScreen.....	225, 231, 232
MenuInflater.....	176	PreferencesManager.....	224
MenuItem.....	165-167, 175, 176	ProgressBar.....	132, 203-205, 208, 209, 212, 214
Message.....	202, 204, 205	Provider.....	342, 352-354, 356-359
MyLocationOverlay.....	408	RadioButton.....	48-50, 53, 58
		RadioGroup.....	48, 49, 53, 58, 60, 61

Keyword Index

RateableWrapper.....120, 121, 123
RateListView.....118, 122, 123
RatingBar.....111, 112, 114, 115, 121, 123, 124
RelativeLayout.....53, 62, 63, 66, 67, 71, 147
Resources.....239, 259
RingtonePreference.....225
RowModel.....114, 115
Runnable.....201, 202, 205-207, 209
ScrollView.....53, 72, 74, 75, 160, 161
SecretsProvider.....350
SecurityException.....364
Service.....370
SharedPreferences.....224, 225, 235
SimpleAdapter.....79
SimpleCursorAdapter.....343, 344, 346
SimplePrefsDemo.....227, 229
SitesOverlay.....405-407
SlidingDrawer.....147, 148, 150
SoftKeyboard.....162
SoftReference.....374
Spanned.....252
Spinner.....84, 85, 92, 96, 343
SQLiteDatabase.....273-275
SQLiteDatabase.CursorFactory.....280
SQLiteOpenHelper.....273
SQLiteQueryBuilder.....276, 278, 279, 353, 354
StackOverflowException.....452, 453
Static.....259
String.....114, 196, 197, 211, 223, 252, 254, 294, 314, 341, 353, 375
TabActivity.....134, 136, 318, 319
TabHost.....132-136, 319
TabHost.TabContentFactory.....138, 139
TabHost.TabSpec.....139
TableLayout.....53, 68-71, 156, 160, 229
TableRow.....68-70
TabSpec.....135, 136
TabView.....144, 319
TabWidget.....133-135, 138, 141, 144
TelephonyManager.....412
TextView.....34, 39-42, 45, 48, 51, 79, 90, 91, 101, 104, 108, 111, 115, 128, 142-144, 160, 163, 181, 344, 420
TextWatcher.....93, 94
TimePicker.....125, 126
TimePickerDialog.....125, 126, 128
Toast.....195, 196, 199, 212, 213, 288, 296, 407
Typeface.....181
Uri.....41, 256, 279, 304, 305, 307, 311, 313, 315, 319, 327, 330, 339-342, 346, 347, 349-358, 360, 361, 382, 412
View.....31, 35, 50, 70, 74, 86, 99, 102-106, 108-111, 115, 121, 124, 138, 139, 141, 166, 173, 175, 196, 201, 206, 318, 433, 435
ViewAnimator.....143
ViewFlipper.....141-146
ViewWrapper.....109-111, 114, 115, 121
Void.....211-213
WeakReference.....374
Weather.....394
WeatherDemo.....296
WeatherPlus.....377, 379, 394
WeatherPlusService.....369, 370, 372, 373, 379
WebKit.....295, 296

Keyword Index

WebSettings.....	192	ACTION_SEARCH.....	423
WebView.....	185-193, 319, 364, 375	ACTION_VIEW.....	394, 313
WebViewClient.....	190, 191	ALTERNATIVE.....	305
XmlPullParser.....	259, 260	CONTENT_URI.....	360
Command.....		DEFAULT.....	305
activityCreator.....	284	DELETE.....	275, 276, 346
adb.....	429	END_DOCUMENT.....	259
adb logcat.....	438	END_TAG.....	259
adb pull.....	281, 439, 447	GET.....	294
adb push.....	281, 439, 447	HORIZONTAL.....	54
adb shell.....	280, 440	INSERT.....	272, 275, 276
android create avd.....	17	INTEGER.....	272
android create project.....	7	LARGER.....	193
ant.....	8, 9	LAUNCHER.....	305, 307
ant jarcore.....	285	LENGTH_LONG.....	196
ant release.....	9	LENGTH_SHORT.....	196
ddms.....	436	MAIN.....	307
dex.....	285	NULL.....	275
hierarchyviewer.....	430, 452	ORDER BY.....	341
jarsigner.....	9, 409	PERMISSION_DENIED.....	367
keytool.....	409	PERMISSION_GRANTED.....	367
mksdcard.....	446, 447	POST.....	294
sqlite3.....	280, 281	R.....	35
Constant.....		RECEIVE_SMS.....	367
ACCESS_COARSE_LOCATION.....	392	RESULT_CANCELLED.....	314
ACCESS_FINE_LOCATION.....	392	RESULT_FIRST_USER.....	314
ACCESS_LOCATION.....	392	RESULT_OK.....	314
ACTION_EDIT.....	304	SELECT.....	272, 276, 279
ACTION_PICK.....	304, 314, 341, 351	SMALLEST.....	193
		START_TAG.....	259, 260

Keyword Index

TEXT.....	259	canGoBackOrForward().....	190
TITLE.....	344	canGoForward().....	189
UPDATE.....	275, 276	check().....	48, 49
VERTICAL.....	54	checkCallingPermission().....	367
WHERE.....	275-277, 279, 341, 347, 353, 355-357	clear().....	224
_id.....	274	clearCache().....	190
Method.....		clearCheck().....	48
add().....	164, 165, 405	clearHistory().....	190
addId().....	340	close().....	150, 243, 274, 279
addIntentOptions().....	165	commit().....	224
addMenu().....	165	create().....	197
addPreferencesFromResource().....	226	createDatabase().....	281
addProximityAlert().....	395	createFromAsset().....	181
addSubMenu().....	165	createItem().....	405
addTab().....	136	createTabContent().....	138
animateClose().....	150	delete().....	275, 276, 346, 356, 357
animateOpen().....	150	doInBackground().....	208, 209, 211, 213
animateToggle().....	150	draw().....	406
appendWhere().....	279	edit().....	224
applyFormat().....	254	execSQL().....	274-276
applyMenuChoice().....	169, 170	execute().....	207, 213, 294
beforeTextChanged().....	94	findViewById()...35, 36, 51, 106, 108-110, 136, 239, 402	
bindView().....	124	finish().....	217, 245
buildForecasts().....	296	generatePage().....	297
buildQuery().....	279	get().....	275
bulkInsert().....	346	getAltitude().....	393
cancel().....	382	getAsInteger().....	275
cancelAll().....	382	getAssets().....	181
canGoBack().....	189	getAsString().....	275
		getAttributeCount().....	261

Keyword Index

getAttributeName()	261	getPhoneType()	412
getBearing()	394	getPosition()	345
getBestProvider()	393	getPreferences()	223, 224
getBoolean()	224	getProgress()	132
getCallState()	412	getProviders()	393
getCheckedItemPositions()	84	getReadableDatabase()	273
getCheckedRadioButtonId()	48	getRequiredColumns()	355
getCollectionType()	358	getResources()	239
getColumnIndex()	279	getRootView()	51
getColumnNames()	279	getSettings()	192
getContentProvider()	346	getSharedPreferences()	223, 224
getContentResolver()	361	getSingleType()	358
getCount()	279	getSpeed()	394
getDefaultSharedPreferences()	224, 225	getString()	251, 252, 254, 279, 345
getFloat()	345	getStringArray()	264
getInputStream()	347	getSubscriberId()	412
getInt()	279, 345	getTag()	109, 114
getIntent()	420	getType()	357, 358
getLastKnownPosition()	393	getView()	79, 90, 102, 103, 105, 106, 110, 114, 118, 120, 121, 124, 344
getLastNonConfigurationInstance()	330	getWritableDatabase()	273
getLatitude()	295	getXml()	259
getListView()	82	goBack()	189
getLongitude()	295	goBackOrForward()	190
getMapController()	402	goForward()	189
getMeMyCurrentLocationNow()	394	handleMessage()	202, 204
getMenuInfo()	167	hasAltitude()	393
getNetworkType()	412	hasBearing()	394
getOutputStream()	347	hasSpeed()	394
getOverlays()	405	incrementProgressBy()	132
getParent()	50	insert()	275, 346, 354, 355, 359

Keyword Index

isAfterLast()	279, 345	notifyChange()	360, 361
isBeforeFirst()	345	notifyMe()	385
isChecked()	45, 48	obtainMessage()	202
isCollectionUri()	355, 356	onActivityResult()	314
isEnabled()	50	onBind()	371
isFirst()	345	onCheckedChanged()	46, 60
isFocused()	50	onClick()	26, 27
isLast()	345	onConfigurationChanged()	331, 333
isNull()	345	onContextItemSelected()	167, 169
isRouteDisplayed()	402	onCreate()	26, 27, 34, 35, 49, 60, 164, 169, 186, 216-219, 230, 242, 254, 273, 296, 327, 328, 333, 343, 351, 352, 370, 372, 420, 423
loadData()	188	onCreateContextMenu()	166, 167, 169
loadTime()	191	onCreateOptionsMenu()	164, 166, 169
loadUrl()	186, 188	onCreatePanelMenu()	165
lock()	150	onDestroy()	217, 370-372
makeMeAnAdapter()	423	onListItemClick()	81, 114
makeText()	196	onLocationChanged()	395
managedQuery()	341-343	onNewIntent()	420, 423
Menu#setGroupCheckable()	165	onOptionsItemSelected()	165-167, 169
MenuItem#setCheckable()	165	onPageStarted()	190
move()	345	onPause()	218, 246, 309, 370, 379, 380, 408
moveToFirst()	279, 345	onPostExecute()	209, 212
moveToLast()	345	onPreExecute()	208
moveToNext()	279, 345	onPrepareOptionsMenu()	164
moveToPosition()	345	onProgressUpdate()	209, 211, 212
moveToPrevious()	345	onRatingBarChanged()	122
newCursor()	280	onRatingChanged()	114
newTabSpec()	135, 136	onReceive()	308
newView()	124	onReceivedHttpRequest()	190
next()	259	onRestart()	217
notify()	382		

Keyword Index

onRestoreInstanceState()	219	removeProximityAlert()	396
onResume()	218, 230, 245, 295, 309, 370, 379, 380, 408	removeUpdates()	395
onRetainNonConfigurationInstance()	330	request()	279, 346
onSaveInstanceState()	217, 219, 325, 327	requestFocus()	50
onSearchRequested()	417, 425	requestLocationUpdates()	394
onStart()	204, 217, 370	restoreMe()	327, 328, 330
onStop()	217	runOnUiThread()	206
onTap()	407	sendBroadcast()	314, 366, 367, 375
onTextChanged()	94	sendMessage()	202
onTooManyRedirects()	190	sendMessageAtFrontOfQueue()	202
onUpgrade()	273	sendMessageAtTime()	202
open()	150	sendMessageDelayed()	202
openFileInput()	243, 246	sendOrderedBroadcast()	314
openFileOutput()	243, 246	set()	288
openRawResource()	239	setAccuracy()	393
populate()	405	setAdapter()	80, 84, 88, 92, 123
populateDefaultValues()	355	setAlphabeticShortcut()	165
populateMenu()	169, 170	setAltitudeRequired()	393
post()	206	setCellRenderer()	78
postDelayed()	206, 379	setCenter()	403
publishProgress()	209, 211, 212	setChecked()	45, 49
query()	276-279, 352-354, 359	setChoiceMode()	82
queryWithFactory()	280	setColumnCollapsed()	71
rawQuery()	276	setColumnShrinkable()	71
rawQueryWithFactory()	280	setColumnStretchable()	71
registerContentObserver()	361	setContent()	135, 136, 138
registerForContextMenu()	166	setContentView()	26, 35, 51
registerReceiver()	309	setCostAllowed()	393
reload()	189	setCurrentTab()	136
remove()	224	setDefaultFontSize()	193

Keyword Index

setDefaultKeyMode()	418	setQwertyMode()	165
setDropDownViewResource()	85	setResult()	314
setDuration()	196	setTag()	109, 110, 115
setEnabled()	50, 176	setText()	27
setFantasyFontFamily()	192	setTextColor()	51
setGravity()	56	setTextSize()	193
setGroupCheckable()	164	setTitle()	197
setGroupEnabled()	176	setTypeface()	32, 181
setGroupVisible()	176	setup()	136
setIcon()	197	setupViews()	333
setImageURI()	41	setUserAgent()	193
setIndeterminate()	132	setView()	196
setIndicator()	135, 136	setVisible()	176
setJavaScriptCanOpenWindowsAutomatically() ()	193	setWebViewClient()	190
setJavaScriptEnabled()	193	setZoom()	403
setLatestEventInfo()	383, 385	shouldOverrideUrlLoading()	190, 191
setListAdapter()	81	show()	196, 197, 199
setMax()	132, 204	showNext()	143
setMessage()	197	size()	405
setNegativeButton()	197	startActivity()	313, 314, 412
setNeutralButton()	197	startActivityForResult()	314
setNumericShortcut()	165	startSearch()	418, 425
setOnClickListener()	26, 139, 245	startService()	370, 378, 379
setOnEditorActionListener()	160	stopService()	378
setOnItemSelectedListener()	80, 84, 88	switch()	166
setOrientation()	54	toggle()	45, 48, 150
setPadding()	56	toggleSatellite()	404
setPositiveButton()	197	toString()	78
setProgress()	132	unlock()	150
setProjectionMap()	279	unregisterContentObserver()	361

Keyword Index

unregisterReceiver().....	309	android:layout_alignTop.....	64, 65
update().....	275, 276, 355-357, 359	android:layout_below.....	64
updateForecast().....	295, 395	android:layout_centerHorizontal.....	63
updateLabel().....	128	android:layout_centerInParent.....	63
updateTime().....	26	android:layout_centerVertical.....	63
valueOf().....	51	android:layout_column.....	69
Property.....		android:layout_gravity.....	56
android:authorities.....	360	android:layout_height.....	33, 55, 65, 134
android:autoText.....	42	android:layout_span.....	69
android:background.....	50	android:layout_toLeftOf.....	64
android:capitalize.....	42	android:layout_toRightOf.....	64
android:collapseColumns.....	71	android:layout_weight.....	55
android:columnWidth.....	88	android:layout_width.....	33, 55, 59, 65
android:completionThreshold.....	92	android:manifest.....	12
android:digits.....	42	android:name.....	14, 359, 364, 372, 425
android:drawSelectorOnTop.....	85, 97	android:nextFocusDown.....	50
android:horizontalSpacing.....	88	android:nextFocusLeft.....	50
android.id.....	33, 34, 48, 63, 133-135	android:nextFocusRight.....	50
android:inputMethod.....	43	android:nextFocusUp.....	50
android:label.....	14	android:numColumns.....	88
android:layout_above.....	64	android:numeric.....	43
android:layout_alignBaseline.....	64	android:orientation.....	54
android:layout_alignBottom.....	64	android:padding.....	56, 57
android:layout_alignLeft.....	64	android:paddingBottom.....	57
android:layout_alignParentBottom.....	63	android:paddingLeft.....	57
android:layout_alignParentLeft.....	63	android:paddingRight.....	57
android:layout_alignParentRight.....	63	android:paddingTop.....	57, 134
android:layout_alignParentTop.....	63, 67	android:password.....	43
android:layout_alignRight.....	64	android:permission.....	366, 373
		android:phoneNumber.....	43

Keyword Index

android:shrinkColumns.....	70	android:text.....	33, 39
android:singleLine.....	42, 43	android:textColor.....	40, 45
android:spacing.....	97	android:textStyle.....	39, 42
android:spinnerSelector.....	97	android:typeface.....	39
android:src.....	41	android:value.....	425
android:stretchColumns.....	70	android:verticalSpacing.....	88
android:stretchMode.....	88	android:visibility.....	50