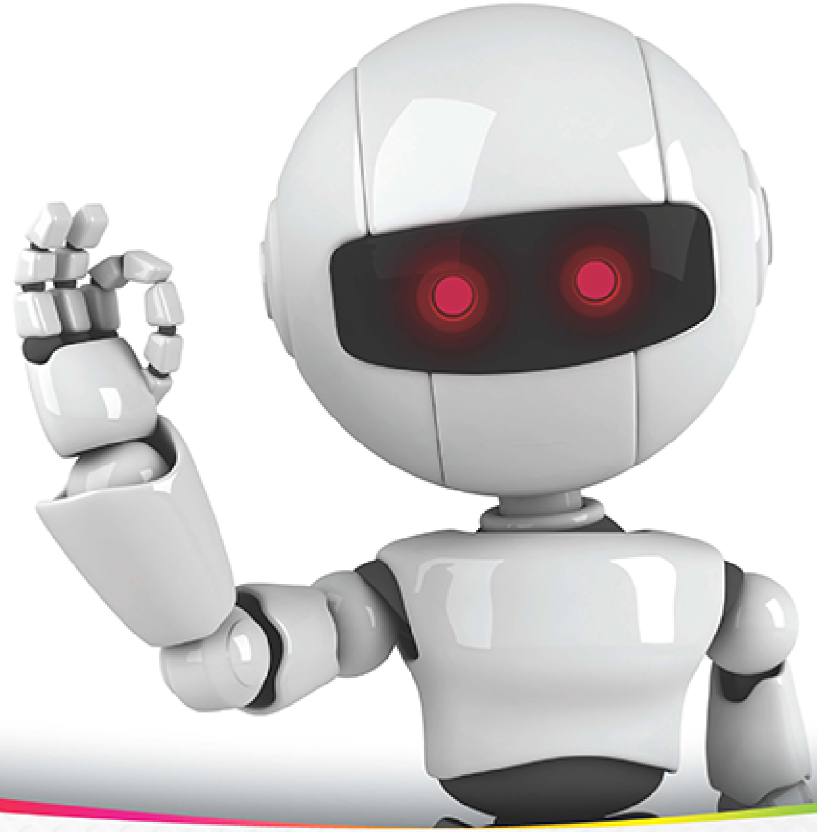
Join the discussion @ p2p.wrox.com

Wrox Programmer to Programmer™



Beginning Android™ Application Development

IN FULL COLOR

Wei-Meng Lee

BEGINNING ANDROID™ APPLICATION DEVELOPMENT

INTRODUCTION xv

CHAPTER 1 Getting Started with Android Programming 1
CHAPTER 2 Activities and Intents 27
CHAPTER 3 Getting to Know the Android User Interface 81
CHAPTER 4 Designing Your User Interface Using Views 125
CHAPTER 5 Displaying Pictures and Menus with Views 169
CHAPTER 6 Data Persistence 203
CHAPTER 7 Content Providers 237
CHAPTER 8 Messaging and Networking 263
CHAPTER 9 Location-Based Services 301
CHAPTER 10 Developing Android Services 331
CHAPTER 11 Publishing Android Applications 359
APPENDIX A Using Eclipse for Android Development 381
APPENDIX B Using the Android Emulator 393
APPENDIX C Answers to Exercises 411

INDEX **415**

BEGINNING Android™ Application Development

Wei-Meng Lee

WILEY

Wiley Publishing, Inc.

Beginning Android™ Application Development

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Copyright © 2011 by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-1-118-01711-1
ISBN: 978-1-118-08729-9 (ebk)
ISBN: 978-1-118-08749-7 (ebk)
ISBN: 978-1-118-08780-0 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Control Number: 2011921777

CONTENTS

INTRODUCTION *xv*

CHAPTER 1: GETTING STARTED WITH ANDROID PROGRAMMING **1**

What Is Android? **2**
Android Versions 2
Features of Android 3
Architecture of Android 3
Android Devices in the Market 4
The Android Market 6
Obtaining the Required Tools **6**
Eclipse 7
Android SDK 7
Android Development Tools (ADT) 7
Creating Android Virtual Devices (AVDs) 11
Creating Your First Android Application 14
Anatomy of an Android Application 22
Summary **25**

CHAPTER 2: ACTIVITIES AND INTENTS **27**

Understanding Activities **27**
Applying Styles and Themes to Activity 32
Hiding the Activity Title 33
Displaying a Dialog Window 34
Displaying a Progress Dialog 39
Linking Activities Using Intents **43**
Resolving Intent Filter Collision 48
Returning Results from an Intent 50
Passing Data Using an Intent Object 54
Calling Built-In Applications Using Intents **56**
Understanding the Intent Object 64
Using Intent Filters 65
Adding Categories 71
Displaying Notifications **73**
Summary **78**

CHAPTER 3: GETTING TO KNOW THE ANDROID USER INTERFACE 81

Understanding the Components of a Screen 81
Views and ViewGroups 82
LinearLayout 83
AbsoluteLayout 87
TableLayout 89
RelativeLayout 91
FrameLayout 93
ScrollView 95
Adapting to Display Orientation 97
Anchoring Views 98
Resizing and Repositioning 101
Managing Changes to Screen Orientation 104
Persisting State Information during Changes in Configuration 108
Detecting Orientation Changes 109
Controlling the Orientation of the Activity 110
Creating the User Interface Programmatically 111
Listening for UI Notifications 114
Overriding Methods Defined in an Activity 114
Registering Events for Views 119
Summary 122

CHAPTER 4: DESIGNING YOUR USER INTERFACE USING VIEWS 125

Basic Views 126
TextView View 126
Button, ImageButton, EditText, CheckBox, ToggleButton, RadioButton, and RadioGroup Views 127
ProgressBar View 135
AutoCompleteTextView View 141
Picker Views 144
TimePicker View 144
Displaying the TimePicker in a Dialog Window 147
DatePicker View 149
Displaying the DatePicker View in a Dialog Window 153
List Views 156
ListView View 156
Customizing the ListView 159
Using the Spinner View 162
Summary 166

CHAPTER 5: DISPLAYING PICTURES AND MENUS WITH VIEWS 169

Using Image Views to Display Pictures 169
Gallery and ImageView Views 170
ImageSwitcher 177
GridView 181
Using Menus with Views 185
Creating the Helper Methods 186
Options Menu 188
Context Menu 190
Some Additional Views 193
AnalogClock and DigitalClock Views 194
WebView 194
Summary 200

CHAPTER 6: DATA PERSISTENCE 203

Saving and Loading User Preferences 203
Using getSharedPreferences() 204
Using getPreferences() 208
Persisting Data to Files 209
Saving to Internal Storage 209
Saving to External Storage (SD Card) 214
Choosing the Best Storage Option 216
Using Static Resources 217
Creating and Using Databases 218
Creating the DBAdapter Helper Class 218
Using the Database Programmatically 224
Adding Contacts 224
Retrieving All the Contacts 225
Retrieving a Single Contact 226
Updating a Contact 227
Deleting a Contact 228
Upgrading the Database 230
Pre-Creating the Database 230
Bundling the Database with an Application 231
Summary 234

CHAPTER 7: CONTENT PROVIDERS 237

Sharing Data in Android 237
Using a Content Provider 238
Predefined Query String Constants 243

Projections 246
Filtering 246
Sorting 247
Creating Your Own Content Providers 247
Using the Content Provider 256
Summary 260

CHAPTER 8: MESSAGING AND NETWORKING 263

SMS Messaging 263
Sending SMS Messages Programmatically 264
Getting Feedback After Sending the Message 267
Sending SMS Messages Using Intent 269
Receiving SMS Messages 270
Updating an Activity from a BroadcastReceiver 273
Invoking an Activity from a BroadcastReceiver 277
Caveats and Warnings 280
Sending E-Mail 281
Networking 284
Downloading Binary Data 286
Downloading Text Files 288
Accessing Web Services 291
Performing Asynchronous Calls 296
Summary 297

CHAPTER 9: LOCATION-BASED SERVICES 301

Displaying Maps 302
Creating the Project 302
Obtaining the Maps API Key 303
Displaying the Map 305
Displaying the Zoom Control 308
Changing Views 310
Navigating to a Specific Location 312
Adding Markers 315
Getting the Location That Was Touched 318
Geocoding and Reverse Geocoding 320
Getting Location Data 322
Monitoring a Location 327
Summary 327

CHAPTER 10: DEVELOPING ANDROID SERVICES 331

Creating Your Own Services 331
- Performing Long-Running Tasks in a Service 336
- Performing Repeated Tasks in a Service 341
- Executing Asynchronous Tasks on Separate Threads Using IntentService 343

Communicating between a Service and an Activity 346
Binding Activities to Services 350
Summary 356

CHAPTER 11: PUBLISHING ANDROID APPLICATIONS 359

Preparing for Publishing 359
- Versioning 360
- Digitally Signing Your Android Applications 362

Deploying APK Files 367
- Using the adb.exe Tool 367
- Using a Web Server 369
- Publishing on the Android Market 372
  - Creating a Developer Profile 372
  - Submitting Your Apps 373

Summary 378

APPENDIX A: USING ECLIPSE FOR ANDROID DEVELOPMENT 381

Getting Around in Eclipse 381
- Workspaces 381
- Package Explorer 382
- Using Projects from Other Workspaces 383
- Editors 385
- Perspectives 387
- Auto Import of Namespaces 387
- Code Completion 388
- Refactoring 388

Debugging 389
- Setting Breakpoints 389
- Exceptions 390

APPENDIX B: USING THE ANDROID EMULATOR 393

Uses of the Android Emulator 393
Installing Custom AVDs 393

Emulating Real Devices 398
SD Card Emulation 399
Emulating Devices with Different Screen Sizes 401
Emulating Physical Capabilities 402
Sending SMS Messages to the Emulator 403
Making Phone Calls 406
Transferring Files into and out of the Emulator 407
Resetting the Emulator 409

APPENDIX C: ANSWERS TO EXERCISES 411

Chapter 1 Answers 411
Chapter 2 Answers 411
Chapter 3 Answers 412
Chapter 4 Answers 412
Chapter 5 Answers 412
Chapter 6 Answers 413
Chapter 7 Answers 413
Chapter 8 Answers 413
Chapter 9 Answers 413
Chapter 10 Answers 414
Chapter 11 Answers 414

INDEX *415*

8 Messaging and Networking

WHAT YOU WILL LEARN IN THIS CHAPTER

- How to send SMS messages programmatically from within your application
- How to send SMS messages using the built-in Messaging application
- How to receive incoming SMS messages
- How to send e-mail messages from your application
- How to connect to the Web using HTTP
- How to consume Web services

Once your basic Android application is up and running, the next interesting thing you can add to it is the capability to communicate with the outside world. You may want your application to send an SMS message to another phone when an event happens (such as when you reach a particular geographical location), or you may wish to access a Web service that provides certain services (such as currency exchange, weather, etc.).

In this chapter, you learn how to send and receive SMS messages programmatically from within your Android application.

You will also learn how to use the HTTP protocol to talk to web servers so that you can download text and binary data. The last part of this chapter shows you how to parse XML files to extract the relevant parts of an XML file — a technique that is useful if you are accessing Web services.

SMS MESSAGING

SMS messaging is one of the main *killer applications* on a mobile phone today — for some users as necessary as the phone itself. Any mobile phone you buy today should have at least SMS messaging capabilities, and nearly all users of any age know how to send and receive such messages. Android comes with a built-in SMS application that enables you to send and receive SMS messages. However, in some cases you might want to integrate SMS capabilities into your own Android application. For example, you might want to write an application that automatically sends a SMS message at regular time intervals. For example, this would be useful if you wanted to track the location of your kids — simply give them an Android device that sends out an SMS message containing its geographical location every 30 minutes. Now you know if they really went to the library after school! (Of course, that would also mean you would have to pay the fees incurred in sending all those SMS messages…)

This section describes how you can programmatically send and receive SMS messages in your Android applications. The good news for Android developers is that you don't need a real device to test SMS messaging: The free Android Emulator provides that capability.

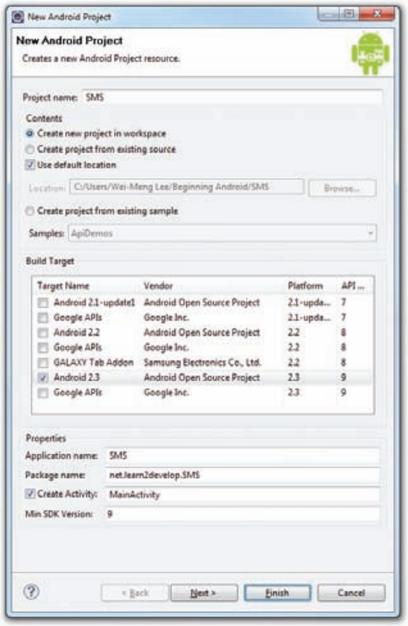
Sending SMS Messages Programmatically

You will first learn how to send SMS messages programmatically from within your application. Using this approach, your application can automatically send an SMS message to a recipient without user intervention. The following Try It Out shows you how.

TRY IT OUT Sending SMS Messages

codefile SMS.zip available for download at Wrox.com

1. Using Eclipse, create a new Android project and name it as shown in Figure 8-1.



Project name: SMS

Contents
- [✓ Create new project in workspace]
- Create project from existing source
- [✓ Use default location]

Location: C:/Users/Wei-Meng Lee/Beginning Android/SMS

- Create project from existing sample

Samples: ApiDemos

Build Target

	Target Name	Vendor	Platform	API ...
	Android 2.1-update1	Android Open Source Project	2.1-upda...	7
	Google APIs	Google Inc.	2.1-upda...	7
	Android 2.2	Android Open Source Project	2.2	8
	Google APIs	Google Inc.	2.2	8
	GALAXY Tab Addon	Samsung Electronics Co., Ltd.	2.2	8
✓	Android 2.3	Android Open Source Project	2.3	9
	Google APIs	Google Inc.	2.3	9

Properties

Application name: SMS

Package name: net.learn2develop.SMS

[✓ Create Activity:] MainActivity

Min SDK Version: 9

FIGURE 8-1

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

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

<Button
    android:id="@+id/btnSendSMS"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Send SMS" />

</LinearLayout>
```

3. In the `AndroidManifest.xml` file, add the following statements in bold:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="net.learn2develop.SMS"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".MainActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="8" />
    <uses-permission android:name="android.permission.SEND_SMS"></uses-permission>
</manifest>
```

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

```
package net.learn2develop.SMS;

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

import android.app.PendingIntent;
import android.content.Intent;
import android.telephony.SmsManager;
import android.view.View;
import android.widget.Button;

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

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

            btnSendSMS = (Button) findViewById(R.id.btnSendSMS);
            btnSendSMS.setOnClickListener(new View.OnClickListener()
            {
                public void onClick(View v)
                {
                    sendSMS("5556", "Hello my friends!");

                }
            });
        }

        //---sends an SMS message to another device---
        private void sendSMS(String phoneNumber, String message)
        {
            SmsManager sms = SmsManager.getDefault();
            sms.sendTextMessage(phoneNumber, null, message, null, null);
        }
    }
```

5. Press F11 to debug the application on the Android Emulator. Using the Android SDK and AVD Manager, launch another AVD.
6. On the first Android Emulator, click the Send SMS button to send an SMS message to the second emulator. The left side of Figure 8-2 shows the SMS message received by the second emulator (note the notification bar at the top of the second emulator).

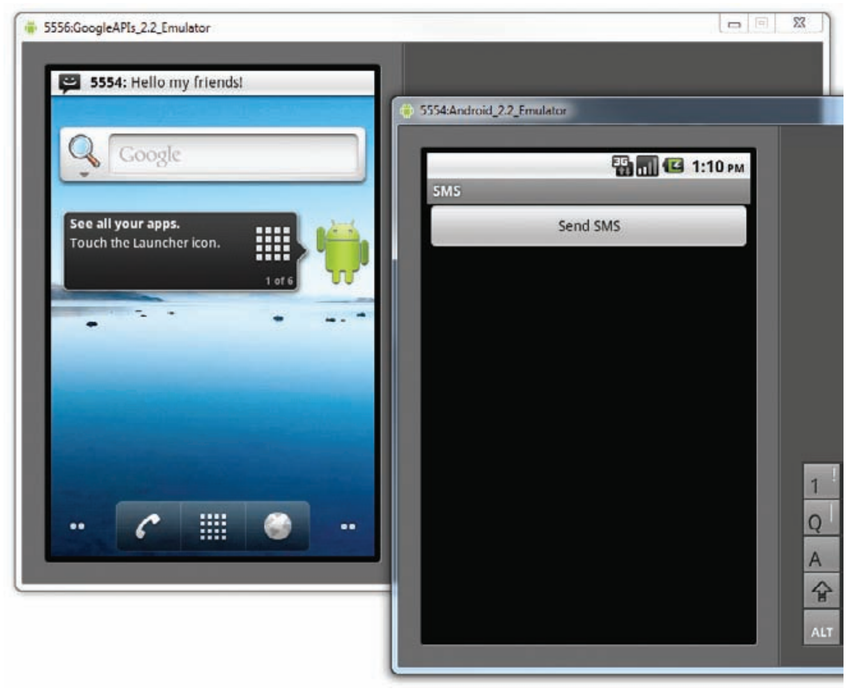


FIGURE 8-2

How It Works

Android uses a permissions-based policy whereby all the permissions needed by an application must be specified in the `AndroidManifest.xml` file. This ensures that when the application is installed, the user knows exactly which access permissions it requires.

Because sending SMS messages incurs additional costs on the user's end, indicating the SMS permissions in the `AndroidManifest.xml` file enables users to decide whether to allow the application to install or not.

To send an SMS message programmatically, you use the `SmsManager` class. Unlike other classes, you do not directly instantiate this class; instead, you call the `getDefault()` static method to obtain a `SmsManager` object. You then send the SMS message using the `sendTextMessage()` method:

```
private void sendSMS(String phoneNumber, String message)
{
    SmsManager sms = SmsManager.getDefault();
    sms.sendTextMessage(phoneNumber, null, message, null, null);
}
```

Following are the five arguments to the `sendTextMessage()` method:

- `destinationAddress` — Phone number of the recipient
- `scAddress` — Service center address; use null for default SMSC
- `text` — Content of the SMS message
- `sentIntent` — Pending intent to invoke when the message is sent (discussed in more detail in the next section)
- `deliveryIntent` — Pending intent to invoke when the message has been delivered (discussed in more detail in the next section)

Getting Feedback after Sending the Message

In the previous section, you learned how to programmatically send SMS messages using the `SmsManager` class; but how do you know that the message has been sent correctly? To do so, you can create two `PendingIntent` objects to monitor the status of the SMS message-sending process. These two `PendingIntent` objects are passed to the last two arguments of the `sendTextMessage()` method. The following code snippets show how you can monitor the status of the SMS message being sent:

```
//---sends an SMS message to another device---
private void sendSMS(String phoneNumber, String message)
{
    String SENT = "SMS_SENT";
    String DELIVERED = "SMS_DELIVERED";

    PendingIntent sentPI = PendingIntent.getBroadcast(this, 0,
        new Intent(SENT), 0);

    PendingIntent deliveredPI = PendingIntent.getBroadcast(this, 0,
            new Intent(DELIVERED), 0);

        //---when the SMS has been sent---
        registerReceiver(new BroadcastReceiver(){
            @Override
            public void onReceive(Context arg0, Intent arg1) {
                switch (getResultCode())
                {
                    case Activity.RESULT_OK:
                        Toast.makeText(getBaseContext(), "SMS sent",
                                Toast.LENGTH_SHORT).show();
                        break;
                    case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
                        Toast.makeText(getBaseContext(), "Generic failure",
                                Toast.LENGTH_SHORT).show();
                        break;
                    case SmsManager.RESULT_ERROR_NO_SERVICE:
                        Toast.makeText(getBaseContext(), "No service",
                                Toast.LENGTH_SHORT).show();
                        break;
                    case SmsManager.RESULT_ERROR_NULL_PDU:
                        Toast.makeText(getBaseContext(), "Null PDU",
                                Toast.LENGTH_SHORT).show();
                        break;
                    case SmsManager.RESULT_ERROR_RADIO_OFF:
                        Toast.makeText(getBaseContext(), "Radio off",
                                Toast.LENGTH_SHORT).show();
                        break;
                }
            }
        }, new IntentFilter(SENT));

        //---when the SMS has been delivered---
        registerReceiver(new BroadcastReceiver(){
            @Override
            public void onReceive(Context arg0, Intent arg1) {
                switch (getResultCode())
                {
                    case Activity.RESULT_OK:
                        Toast.makeText(getBaseContext(), "SMS delivered",
                                Toast.LENGTH_SHORT).show();
                        break;
                    case Activity.RESULT_CANCELED:
                        Toast.makeText(getBaseContext(), "SMS not delivered",
                                Toast.LENGTH_SHORT).show();
                        break;
                }
            }
        }, new IntentFilter(DELIVERED));

        SmsManager sms = SmsManager.getDefault();
        sms.sendTextMessage(phoneNumber, null, message, sentPI, deliveredPI);
    }
```

Here, you created two PendingIntent objects. You then registered for two BroadcastReceivers. These two BroadcastReceivers listen for intents that match "SMS_SENT" and "SMS_DELIVERED" (which are fired by the OS when the message has been sent and delivered, respectively). Within each BroadcastReceiver you override the onReceive() method and get the current result code.

The two PendingIntent objects are passed into the last two arguments of the sendTextMessage() method:

```
sms.sendTextMessage(phoneNumber, null, message, sentPI, deliveredPI);
```

In this case, whether a message has been sent correctly or failed to be delivered, you will be notified of its status via the two PendingIntent objects.

Sending SMS Messages Using Intent

Using the SmsManager class, you can send SMS messages from within your application without the need to involve the built-in Messaging application. However, sometimes it would be easier if you could simply invoke the built-in Messaging application and let it do all the work of sending the message.

To activate the built-in Messaging application from within your application, you can use an Intent object together with the MIME type "vnd.android-dir/mms-sms" as shown by the following code snippet:

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

    btnSendSMS = (Button) findViewById(R.id.btnSendSMS);
    btnSendSMS.setOnClickListener(new View.OnClickListener()
    {
        public void onClick(View v)
        {
            //sendSMS("5556", "Hello my friends!");
            Intent i = new
                Intent(android.content.Intent.ACTION_VIEW);
            i.putExtra("address", "5556; 5558; 5560");

            i.putExtra("sms_body", "Hello my friends!");
            i.setType("vnd.android-dir/mms-sms");
            startActivity(i);
        }
    });
}
```

This will invoke the Messaging application, as shown in Figure 8-3. Note that you can send your SMS to multiple recipients by simply separating each phone number with a semicolon (in the putExtra() method).

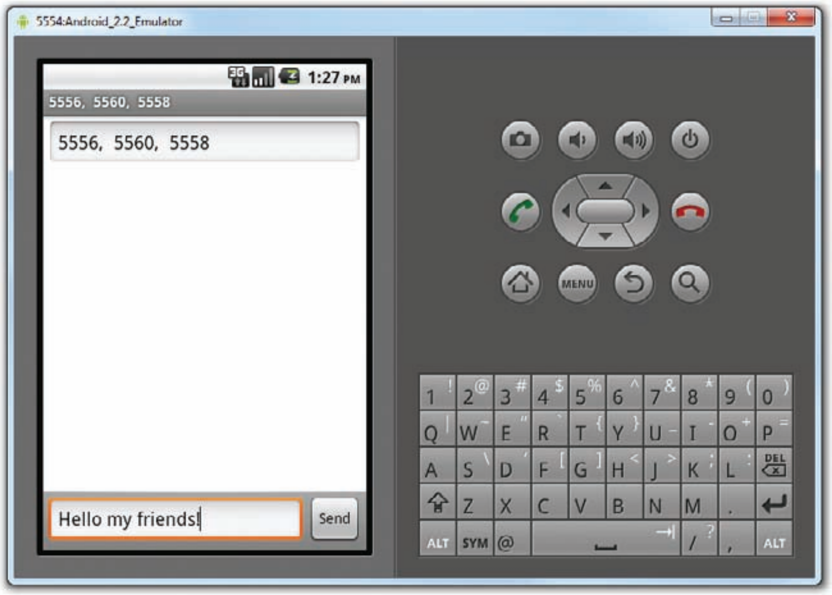


FIGURE 8-3

NOTE *If you use this method to invoke the Messaging application, there is no need to ask for the* `SMS_SEND` *permission in* `AndroidManifest.xml` *because your application is ultimately not the one sending the message.*

Receiving SMS Messages

Besides sending SMS messages from your Android applications, you can also receive incoming SMS messages from within your application by using a `BroadcastReceiver` object. This is useful when you want your application to perform an action when a certain SMS message is received. For example, you might want to track the location of your phone in case it is lost or stolen. In this case, you can write an application that automatically listens for SMS messages containing some secret code. Once that message is received, you can then send an SMS message containing the location's coordinates back to the sender.

The following Try It Out shows how to programmatically listen for incoming SMS messages.

TRY IT OUT Receiving SMS Messages

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

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

```
        package="net.learn2develop.SMS"
        android:versionCode="1"
        android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".MainActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name=".SMSReceiver">
            <intent-filter>
                <action android:name=
                    "android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>
    </application>
    <uses-sdk android:minSdkVersion="8" />
    <uses-permission android:name="android.permission.SEND_SMS"></uses-permission>
    <uses-permission android:name="android.permission.RECEIVE_SMS">
    </uses-permission>
</manifest>
```

2. In the `src` folder of the project, add a new Class file to the package name and call it `SMSReceiver.java` (see Figure 8-4).

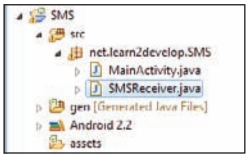


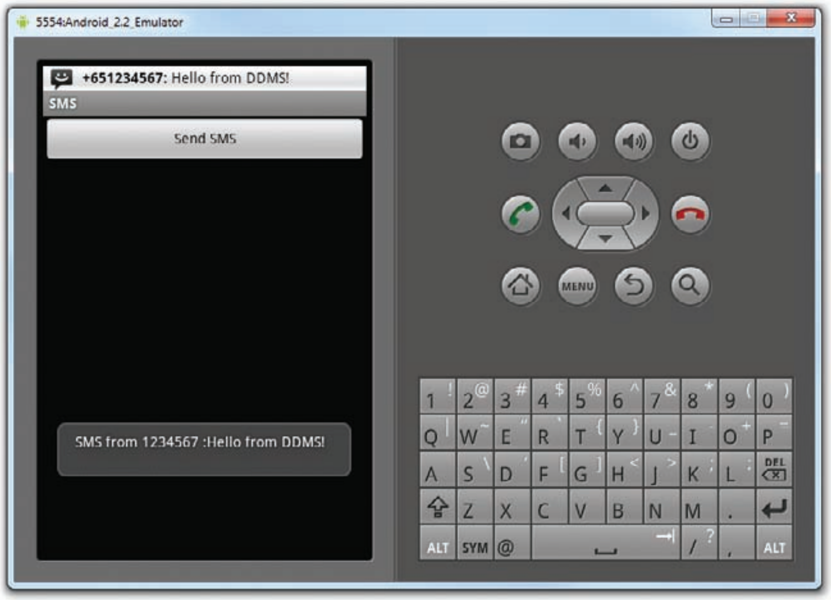
FIGURE 8-4

3. Code the `SMSReceiver.java` file as follows:

```
package net.learn2develop.SMS;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsMessage;
import android.widget.Toast;

public class SMSReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        //---get the SMS message passed in---
        Bundle bundle = intent.getExtras();
        SmsMessage[] msgs = null;
        String str = "";
        if (bundle != null)
        {
            //---retrieve the SMS message received---
            Object[] pdus = (Object[]) bundle.get("pdus");
            msgs = new SmsMessage[pdus.length];
            for (int i=0; i<msgs.length; i++){
                msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
                str += "SMS from " + msgs[i].getOriginatingAddress();
```

```
                    str += " :";
                    str += msgs[i].getMessageBody().toString();
                    str += "\n";
                }
                //---display the new SMS message---
                Toast.makeText(context, str, Toast.LENGTH_SHORT).show();
            }
        }
    }
```

4. Press F11 to debug the application on the Android Emulator.
5. Using the DDMS, send a message to the emulator. Your application should be able to receive the message and display it using the `Toast` class (see Figure 8-5).



+651234567: Hello from DDMS!

SMS from 1234567 :Hello from DDMS!

FIGURE 8-5

How It Works

To listen for incoming SMS messages, you create a `BroadcastReceiver` class. The `BroadcastReceiver` class enables your application to receive intents sent by other applications using the `sendBroadcast()` method. Essentially, it enables your application to handle events raised by other applications. When an intent is received, the `onReceive()` method is called; hence, you need to override this.

When an incoming SMS message is received, the `onReceive()` method is fired. The SMS message is contained in the `Intent` object (intent; the second parameter in the `onReceive()` method) via a `Bundle` object. The messages are stored in an `Object` array in the PDU format. To extract each message, you use the static `createFromPdu()` method from the `SmsMessage` class. The SMS message is then displayed using the `Toast` class. The phone number of the sender is obtained via the `getOriginatingAddress()`

method, so if you need to send an autoreply to the sender, this is the method to obtain the sender's phone number.

One interesting characteristic of the `BroadcastReceiver` is that you can continue to listen for incoming SMS messages even if the application is not running; as long as the application is installed on the device, any incoming SMS messages will be received by the application.

Updating an Activity from a BroadcastReceiver

The previous section described how you can use a `BroadcastReceiver` class to listen for incoming SMS messages and then use the `Toast` class to display the received SMS message. Often, you'll want to send the SMS message back to the main activity of your application. For example, you might wish to display the message in a `TextView`. The following Try It Out demonstrates how you can do this.

TRY IT OUT Creating a View-Based Application Project

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

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

<Button
    android:id="@+id/btnSendSMS"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Send SMS" />

<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

</LinearLayout>
```

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

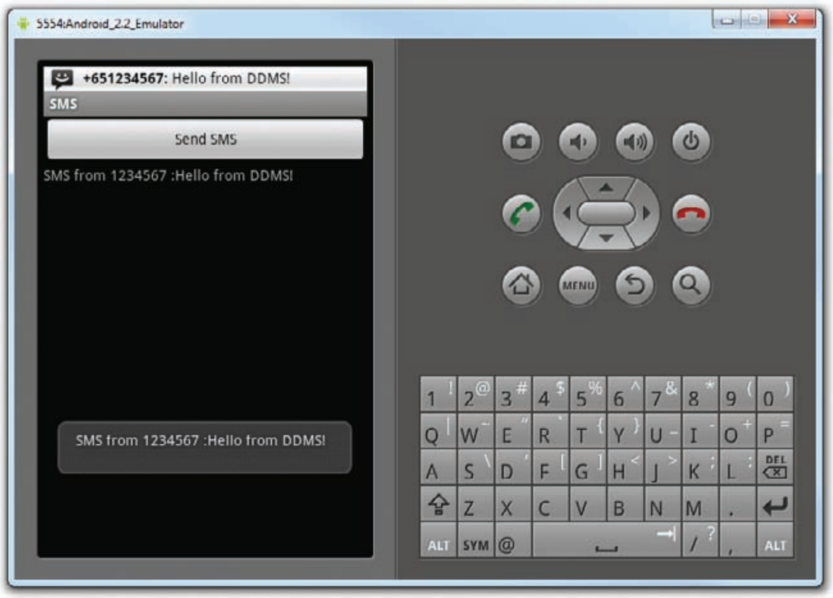
```
package net.learn2develop.SMS;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsMessage;
import android.widget.Toast;

public class SMSReceiver extends BroadcastReceiver
```

```
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        //---get the SMS message passed in---
        Bundle bundle = intent.getExtras();
        SmsMessage[] msgs = null;
        String str = "";
        if (bundle != null)
        {
            //---retrieve the SMS message received---
            Object[] pdus = (Object[]) bundle.get("pdus");
            msgs = new SmsMessage[pdus.length];
            for (int i=0; i<msgs.length; i++){
                msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
                str += "SMS from " + msgs[i].getOriginatingAddress();
                str += " :";
                str += msgs[i].getMessageBody().toString();
                str += "\n";
            }
            //---display the new SMS message---
            Toast.makeText(context, str, Toast.LENGTH_SHORT).show();

            //---send a broadcast intent to update the SMS received in the activity---
            Intent broadcastIntent = new Intent();
            broadcastIntent.setAction("SMS_RECEIVED_ACTION");
            broadcastIntent.putExtra("sms", str);
            context.sendBroadcast(broadcastIntent);
        }
    }
}
```

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

```
package net.learn2develop.SMS;

import android.app.Activity;
import android.os.Bundle;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.telephony.SmsManager;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

import android.content.BroadcastReceiver;
import android.content.IntentFilter;
import android.widget.TextView;

public class MainActivity extends Activity {
    Button btnSendSMS;
    IntentFilter intentFilter;

    private BroadcastReceiver intentReceiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                //---display the SMS received in the TextView---
                TextView SMSes = (TextView) findViewById(R.id.textView1);
                SMSes.setText(intent.getExtras().getString("sms"));
            }
        };

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

            //---intent to filter for SMS messages received---
            intentFilter = new IntentFilter();
            intentFilter.addAction("SMS_RECEIVED_ACTION");

            btnSendSMS = (Button) findViewById(R.id.btnSendSMS);
            btnSendSMS.setOnClickListener(new View.OnClickListener()
            {
                public void onClick(View v)
                {
                    //sendSMS("5554", "Hello my friends!");

                    Intent i = new
                        Intent(android.content.Intent.ACTION_VIEW);
                    i.putExtra("address", "5556; 5558; 5560");
                    i.putExtra("sms_body", "Hello my friends!");
                    i.setType("vnd.android-dir/mms-sms");
                    startActivity(i);
                }
            });
        }

        @Override
        protected void onResume() {
            //---register the receiver---
            registerReceiver(intentReceiver, intentFilter);
            super.onResume();
        }

        @Override
        protected void onPause() {
            //---unregister the receiver---
            unregisterReceiver(intentReceiver);
            super.onPause();
        }

        //---sends an SMS message to another device---
        private void sendSMS(String phoneNumber, String message)
        {
            //...
        }
    }
```

4. Press F11 to debug the application on the Android Emulator. Using the DDMS, send an SMS message to the emulator. Figure 8-6 shows the `Toast` class displaying the message received, and the `TextView` showing the message received.



SMS from 1234567 :Hello from DDMS!

SMS from 1234567 :Hello from DDMS!

FIGURE 8-6

How It Works

You first added a `TextView` to your activity so that it can be used to display the received SMS message.

Next, you modified the `SMSReceiver` class so that when it receives an SMS message, it will broadcast another `Intent` object so that any applications listening for this intent can be notified (which we will implement in the activity next). The SMS received is also sent out via this intent:

```
//---send a broadcast intent to update the SMS received in the activity---
Intent broadcastIntent = new Intent();
broadcastIntent.setAction("SMS_RECEIVED_ACTION");
broadcastIntent.putExtra("sms", str);
context.sendBroadcast(broadcastIntent);
```

Next, in your activity you created a `BroadcastReceiver` object to listen for broadcast intents:

```
private BroadcastReceiver intentReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        //---display the SMS received in the TextView---
        TextView SMSes = (TextView) findViewById(R.id.textView1);
        SMSes.setText(intent.getExtras().getString("sms"));
    }
};
```

When a broadcast intent is received, you update the SMS message in the `TextView`.

You need to create an `IntentFilter` object so that you can listen for a particular intent. In this case, the intent is `"SMS_RECEIVED_ACTION"`:

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

    //---intent to filter for SMS messages received---
    intentFilter = new IntentFilter();
    intentFilter.addAction("SMS_RECEIVED_ACTION");
    //...
}
```

Finally, you register the `BroadcastReceiver` in the activity's `onResume()` event and unregister it in the `onPause()` event:

```
@Override
protected void onResume() {
    //---register the receiver---
    registerReceiver(intentReceiver, intentFilter);
    super.onResume();
}

@Override
protected void onPause() {
    //---unregister the receiver---
    unregisterReceiver(intentReceiver);
    super.onPause();
}
```

This means that the `TextView` will display the SMS message only when the message is received while the activity is visible on the screen. If the SMS message is received when the activity is not in the foreground, the `TextView` will not be updated.

Invoking an Activity from a BroadcastReceiver

The previous example shows how you can pass the SMS message received to be displayed in the activity. However, in many situations your activity may be in the background when the SMS message is received. In this case, it would be useful to be able to bring the activity to the foreground when a message is received. The following Try It Out shows you how.

TRY IT OUT Invoking an Activity

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

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

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //---intent to filter for SMS messages received---
        intentFilter = new IntentFilter();
        intentFilter.addAction("SMS_RECEIVED_ACTION");

        //---register the receiver---
        registerReceiver(intentReceiver, intentFilter);

        btnSendSMS = (Button) findViewById(R.id.btnSendSMS);
        btnSendSMS.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View v)
            {
                //sendSMS("5554", "Hello my friends!");
                Intent i = new
                    Intent(android.content.Intent.ACTION_VIEW);
                i.putExtra("address", "5556; 5558; 5560");


                i.putExtra("sms_body", "Hello my friends!");
                i.setType("vnd.android-dir/mms-sms");
                startActivity(i);
            }
        });
    }

    @Override
    protected void onResume() {
        //---register the receiver---
        //registerReceiver(intentReceiver, intentFilter);
        super.onResume();
    }

    @Override
    protected void onPause() {
        //---unregister the receiver---
        //unregisterReceiver(intentReceiver);
        super.onPause();
    }

    @Override
    protected void onDestroy() {
        //---unregister the receiver---
        unregisterReceiver(intentReceiver);
        super.onPause();
    }
```

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

```
    @Override
    public void onReceive(Context context, Intent intent)
    {
            //---get the SMS message passed in---
            Bundle bundle = intent.getExtras();
            SmsMessage[] msgs = null;
            String str = "";
            if (bundle != null)
            {
                //---retrieve the SMS message received---
                Object[] pdus = (Object[]) bundle.get("pdus");
                msgs = new SmsMessage[pdus.length];
                for (int i=0; i<msgs.length; i++){
                    msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
                    str += "SMS from " + msgs[i].getOriginatingAddress();
                    str += " :";
                    str += msgs[i].getMessageBody().toString();
                    str += "\n";
                }
                //---display the new SMS message---
                Toast.makeText(context, str, Toast.LENGTH_SHORT).show();

                //---launch the MainActivity---
                Intent mainActivityIntent = new Intent(context, MainActivity.class);
                mainActivityIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                context.startActivity(mainActivityIntent);

                //---send a broadcast to update the SMS received in the activity---
                Intent broadcastIntent = new Intent();
                broadcastIntent.setAction("SMS_RECEIVED_ACTION");
                broadcastIntent.putExtra("sms", str);
                context.sendBroadcast(broadcastIntent);
            }
        }
```

3. Modify the `main.xml` file as follows:

```
        <activity android:name=".MainActivity"
                  android:label="@string/app_name"
                  android:launchMode="singleTask" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

4. Press F11 to debug the application on the Android Emulator. When the `MainActivity` is shown, click the Home button to send the activity to the background.

5. Use the DDMS to send an SMS message to the emulator again. This time, note that the activity will be brought to the foreground, displaying the SMS message received.

How It Works

In the `MainActivity` class, you first register the `BroadcastReceiver` in the activity's `onCreate()` event, instead of the `onResume()` event; and instead of unregistering it in the `onPause()` event, you now unregister it in the `onDestroy()` event. This ensures that even if the activity is in the background, it will still be able to listen for the broadcast intent.

Next, you modify the `onReceive()` event in the `SMSReceiver` class by using an intent to bring the activity to the foreground before broadcasting another intent:

```
//---launch the MainActivity---
Intent mainActivityIntent = new Intent(context, MainActivity.class);
mainActivityIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
context.startActivity(mainActivityIntent);

//---send a broadcast to update the SMS received in the activity---
Intent broadcastIntent = new Intent();
broadcastIntent.setAction("SMS_RECEIVED_ACTION");
broadcastIntent.putExtra("sms", str);
context.sendBroadcast(broadcastIntent);
```

The `startActivity()` method launches the activity and brings it to the foreground. Note that you need to set the `Intent.FLAG_ACTIVITY_NEW_TASK` flag because calling `startActivity()` from outside of an activity context requires the `FLAG_ACTIVITY_NEW_TASK` flag.

You also need to set the `launchMode` attribute of the `<activity>` element in the `AndroidManifest.xml` file to `singleTask`:

```
<activity android:name=".MainActivity"
          android:label="@string/app_name"
          android:launchMode="singleTask" >
```

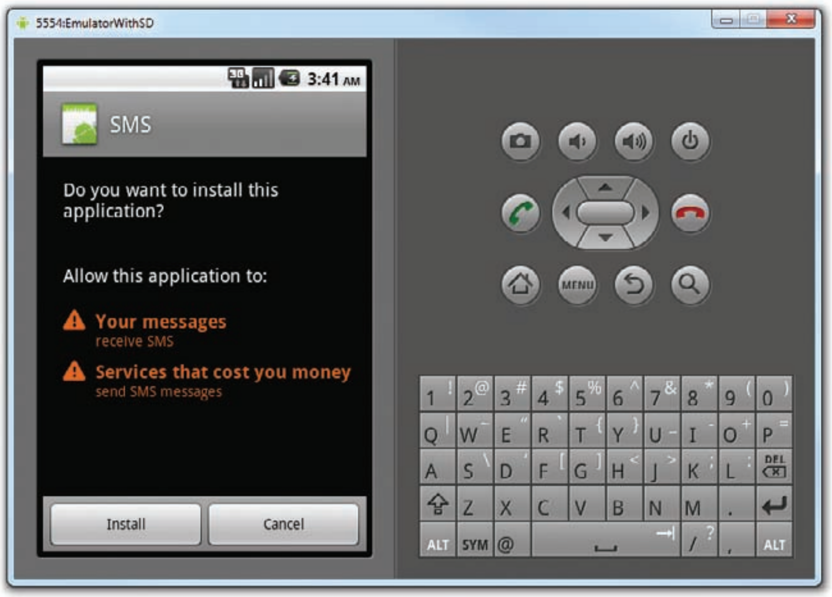
If you don't set this, multiple instances of the activity will be launched as your application receives SMS messages.

Note that in this example, when the activity is in the background (such as when you click the Home button to show the home screen), the activity is brought to the foreground and its `TextView` is updated with the SMS received. However, if the activity was killed (such as when you click the Back button to destroy it), the activity is launched again but the `TextView` is not updated.

Caveats and Warnings

While the ability to send and receive SMS messages makes Android a very compelling platform for developing sophisticated applications, this flexibility comes with a price. A seemingly innocent application may send SMS messages behind the scene without the user knowing, as demonstrated by a recent case of an SMS-based Trojan Android application (`http://forum.vodafone.co.nz/topic/5719-android-sms-trojan-warning/`). Claiming to be a media player, once installed, the application sends SMS messages to a premium number, resulting in huge phone bills for the user.

While the user needs to explicitly give permission to your application, the request for permission is only shown at installation time. Figure 8-7 shows the request for permission that appears when you try to install the application (as an APK file; Chapter 11 discusses packaging your Android applications in more detail) on the emulator (same as on a real device). If the user clicks the Install button, he or she is considered to have given permission to allow the application to send and receive SMS messages. This is dangerous, as after the application is installed it can send and receive SMS messages without ever prompting the user again.



Do you want to install this application?

Allow this application to:

Your messages
receive SMS

Services that cost you money
send SMS messages

FIGURE 8-7

In addition to this, the application can also "sniff" for incoming SMS messages. For example, based on the techniques you learned from the previous section, you can easily write an application that checks for certain keywords in the SMS message. When an SMS message contains the keyword you are looking for, you can then use the Location Manager (discussed in Chapter 9) to obtain your geographical location and then send the coordinates back to the sender of the SMS message. The sender could then easily track your location. All these tasks can be done easily without the user knowing it! That said, users should try to avoid installing Android applications that come from dubious sources, such as from unknown websites, strangers, etc.

SENDING E-MAIL

Like SMS messaging, Android also supports e-mail. The Gmail/Email application on Android enables you to configure an e-mail account using POP3 or IMAP. Besides sending and receiving e-mails using the Gmail/Email application, you can also send e-mail messages programmatically from within your Android application. The following Try It Out shows you how.

TRY IT OUT Sending E-mail Programmatically

codefile Emails.zip available for download at Wrox.com

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

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

<Button
    android:id="@+id/btnSendEmail"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Send Email" />

</LinearLayout>
```

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

```
package net.learn2develop.Email;

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

import android.content.Intent;
import android.net.Uri;
import android.view.View;
import android.widget.Button;

public class MainActivity extends Activity {
    Button btnSendEmail;

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

        btnSendEmail = (Button) findViewById(R.id.btnSendEmail);
        btnSendEmail.setOnClickListener(new View.OnClickListener()
        {
            public void onClick(View v)
            {
                String[] to = {"weimenglee@learn2develop.net", "weimenglee@gmail.com"};
                String[] cc = {"course@learn2develop.net"};
                sendEmail(to, cc, "Hello", "Hello my friends!");
            }
        });
        }

        //---sends an SMS message to another device---
        private void sendEmail(String[] emailAddresses, String[] carbonCopies,
        String subject, String message)
        {
            Intent emailIntent = new Intent(Intent.ACTION_SEND);
            emailIntent.setData(Uri.parse("mailto:"));
            String[] to = emailAddresses;
            String[] cc = carbonCopies;
            emailIntent.putExtra(Intent.EXTRA_EMAIL, to);
            emailIntent.putExtra(Intent.EXTRA_CC, cc);
            emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject);
            emailIntent.putExtra(Intent.EXTRA_TEXT, message);
            emailIntent.setType("message/rfc822");
            startActivity(Intent.createChooser(emailIntent, "Email"));
        }
    }
```

4. Press F11 to test the application on a real Android device. Click the Send Email button and you should see the Email application launched in your device, as shown in Figure 8-8.

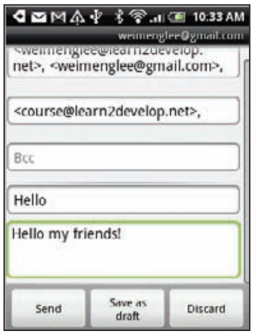

10:33 AM
weimenglee@gmail.com
<weimenglee@learn2develop.net>, <weimenglee@gmail.com>,
<course@learn2develop.net>,
Bcc
Hello
Hello my friends!

FIGURE 8-8

How It Works

In this example, you are launching the built-in Email application to send an e-mail message. To do so, you use an `Intent` object and set the various parameters using the `setData()`, `putExtra()`, and `setType()` methods:

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);
emailIntent.setData(Uri.parse("mailto:"));
String[] to = emailAddresses;
String[] cc = carbonCopies;
emailIntent.putExtra(Intent.EXTRA_EMAIL, to);
emailIntent.putExtra(Intent.EXTRA_CC, cc);
emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject);
emailIntent.putExtra(Intent.EXTRA_TEXT, message);
emailIntent.setType("message/rfc822");
startActivity(Intent.createChooser(emailIntent, "Email"));
```

NETWORKING

The previous sections covered how to get connected to the outside world using SMS and e-mail. Another way to achieve that is to use the HTTP protocol. Using the HTTP protocol, you can perform a wide variety of tasks, such as downloading web pages from a web server, downloading binary data, and so on.

The following Try It Out creates an Android project so that you can use the HTTP protocol to connect to the Web to download all sorts of data.

TRY IT OUT Creating the Project

codefile Networking.zip available for download at Wrox.com

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

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="net.learn2develop.Networking"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".MainActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="8" />
    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
</manifest>
```

3. Import the following namespaces in the `MainActivity.java` file:

```
package net.learn2develop.Networking;

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

import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.widget.ImageView;
import android.widget.Toast;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

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

4. Define the `OpenHttpConnection()` method in the `MainActivity.java` file:

```
public class MainActivity extends Activity {

    private InputStream OpenHttpConnection(String urlString)
    throws IOException
    {
        InputStream in = null;
        int response = -1;

        URL url = new URL(urlString);
        URLConnection conn = url.openConnection();

        if (!(conn instanceof HttpURLConnection))
            throw new IOException("Not an HTTP connection");
        try{
            HttpURLConnection httpConn = (HttpURLConnection) conn;
            httpConn.setAllowUserInteraction(false);
            httpConn.setInstanceFollowRedirects(true);
            httpConn.setRequestMethod("GET");
            httpConn.connect();
            response = httpConn.getResponseCode();
            if (response == HttpURLConnection.HTTP_OK) {
                in = httpConn.getInputStream();
            }
        }
        catch (Exception ex)
        {
            throw new IOException("Error connecting");
        }
        return in;
    }

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

How It Works

Because you are using the HTTP protocol to connect to the Web, your application needs the `INTERNET` permission; hence, the first thing you do is add the permission in the `AndroidManifest.xml` file.

You then define the `OpenHttpConnection()` method, which takes a URL string and returns an `InputStream` object. Using an `InputStream` object, you can download the data by reading bytes from the stream object. In this method, you made use of the `HttpURLConnection` object to open an HTTP connection with a remote URL. You set all the various properties of the connection, such as the request method, and so on:

```
HttpURLConnection httpConn = (HttpURLConnection) conn;
httpConn.setAllowUserInteraction(false);
httpConn.setInstanceFollowRedirects(true);
httpConn.setRequestMethod("GET");
```

After you try to establish a connection with the server, you get the HTTP response code from it. If the connection is established (via the response code `HTTP_OK`), then you proceed to get an `InputStream` object from the connection:

```
httpConn.connect();
response = httpConn.getResponseCode();
if (response == HttpURLConnection.HTTP_OK) {
    in = httpConn.getInputStream();
}
```

Using the `InputStream` object, you can then start to download the data from the server.

Downloading Binary Data

One of the common tasks you need to perform is downloading binary data from the Web. For example, you may want to download an image from a server so that you can display it in your application. The following Try It Out shows how this is done.

TRY IT OUT Creating the Project

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

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

<ImageView
    android:id="@+id/img"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center" />

</LinearLayout>
```

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

```
public class MainActivity extends Activity {
    ImageView img;

    private InputStream OpenHttpConnection(String urlString)
    throws IOException
    {
        //...
    }

    private Bitmap DownloadImage(String URL)
    {
        Bitmap bitmap = null;
        InputStream in = null;
        try {
            in = OpenHttpConnection(URL);
            bitmap = BitmapFactory.decodeStream(in);
            in.close();
        } catch (IOException e1) {
            Toast.makeText(this, e1.getLocalizedMessage(),
                Toast.LENGTH_LONG).show();

            e1.printStackTrace();
        }
        return bitmap;
    }

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

        //---download an image---
        Bitmap bitmap =
            DownloadImage(
            "http://www.streetcar.org/mim/cable/images/cable-01.jpg");
        img = (ImageView) findViewById(R.id.img);
        img.setImageBitmap(bitmap);
    }
}
```

3. Press F11 to debug the application on the Android Emulator. Figure 8-9 shows the image downloaded from the Web and then displayed in the `ImageView`.

How It Works

The `DownloadImage()` method takes the URL of the image to download and then opens the connection to the server using the `OpenHttpConnection()` method that you have defined earlier. Using the `InputStream` object returned by the connection, the `decodeStream()` method from the `BitmapFactory` class is used to download and decode the data into a `Bitmap` object. The `DownloadImage()` method returns a `Bitmap` object.

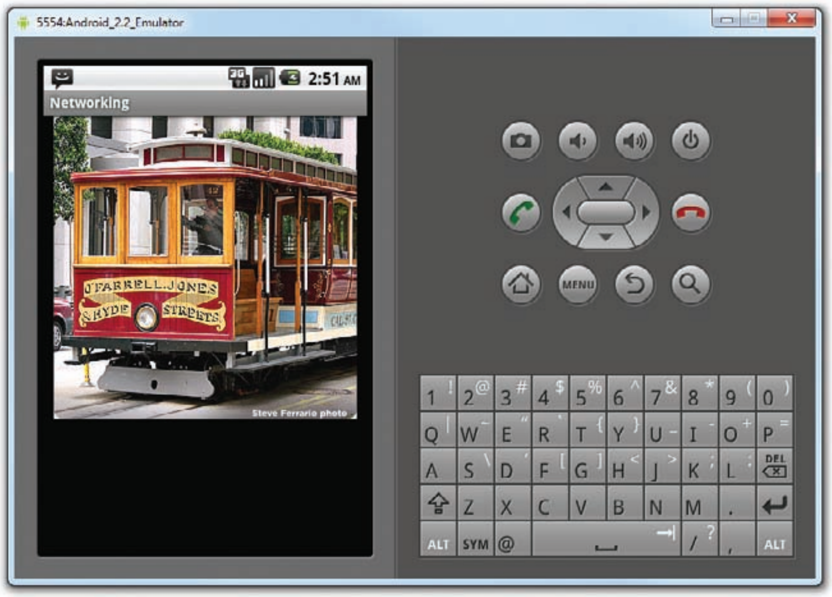


FIGURE 8-9

The image is then displayed using an `ImageView` view.

REFERRING TO LOCALHOST FROM YOUR EMULATOR

When working with the Android Emulator, you may frequently need to access data hosted on the local web server using `localhost`. For example, your own Web services is likely to be hosted on your local computer during development time and you want to test it on the same development machine you use to write your Android applications. In such cases, you should use the special IP address of 10.0.2.2 (not 127.0.0.1) to refer to the host computer's loopback interface. From the Android Emulator's perspective, `localhost` (127.0.0.1) refers to its own loopback interface.

Downloading Text Files

Besides downloading binary data, you can also download plain-text files. For example, you might be writing an RSS Reader application and hence need to download RSS XML feeds for processing. The following Try It Out shows how you can download a plain-text file in your application.

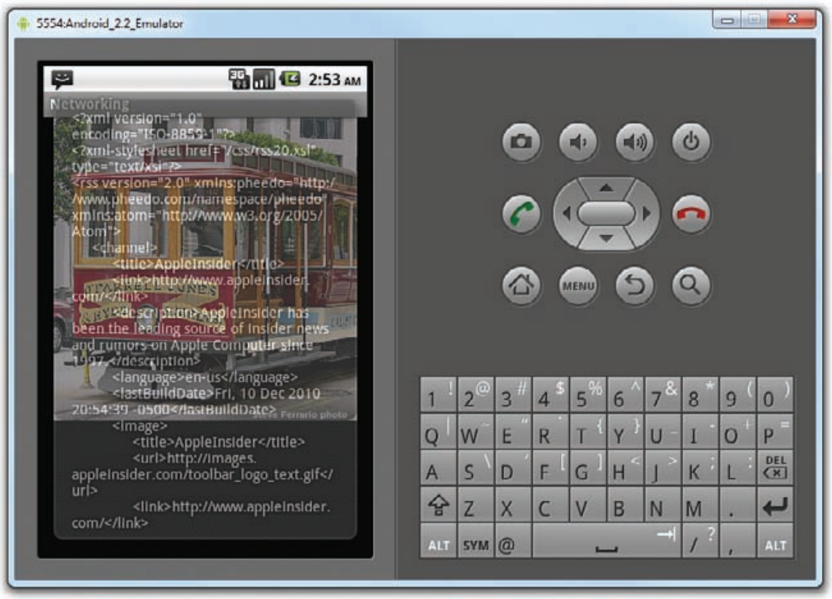
TRY IT OUT Downloading Plain-Text Files

1. Using the same project created earlier, add the following statements in bold to the `MainActivity.java` file:

```
public class MainActivity extends Activity {
    ImageView img;

    private InputStream OpenHttpConnection(String urlString)
    throws IOException
    {
        //...
    }

    private Bitmap DownloadImage(String URL)
    {
        //...
    }

    private String DownloadText(String URL)
    {
        int BUFFER_SIZE = 2000;
        InputStream in = null;
        try {
            in = OpenHttpConnection(URL);
        } catch (IOException e1) {
            Toast.makeText(this, e1.getLocalizedMessage(),
                Toast.LENGTH_LONG).show();

            e1.printStackTrace();
            return "";
        }

        InputStreamReader isr = new InputStreamReader(in);
        int charRead;
        String str = "";
        char[] inputBuffer = new char[BUFFER_SIZE];
        try {
            while ((charRead = isr.read(inputBuffer))>0)
            {
                //---convert the chars to a String---
                String readString =
                    String.copyValueOf(inputBuffer, 0, charRead);
                str += readString;
                inputBuffer = new char[BUFFER_SIZE];
            }
            in.close();
        } catch (IOException e) {
            Toast.makeText(this, e.getLocalizedMessage(),
                Toast.LENGTH_LONG).show();

            e.printStackTrace();
```

```
                return "";
            }
            return str;
        }

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

            //---download an image---
            Bitmap bitmap =
                DownloadImage(
                "http://www.streetcar.org/mim/cable/images/cable-01.jpg");
            img = (ImageView) findViewById(R.id.img);
            img.setImageBitmap(bitmap);

            //---download an RSS feed---
            String str = DownloadText(
                "http://www.appleinsider.com/appleinsider.rss");
            Toast.makeText(getBaseContext(), str,
                                Toast.LENGTH_SHORT).show();
        }
    }
```

2. Press F11 to debug the application on the Android Emulator. Figure 8-10 shows the RSS feed downloaded and displayed using the `Toast` class.



```
<?xml version="1.0"
encoding="ISO-8859-1"?>
<?xml-stylesheet href="/css/rss20.xsl"
type="text/xsl"?>
<rss version="2.0" xmlns:pheedo="http:/
/www.pheedo.com/namespace/pheedo"
xmlns:atom="http://www.w3.org/2005/
Atom">
    <channel>
        <title>AppleInsider</title>
        <link>http://www.appleinsider.
com/</link>
        <description>AppleInsider has
been the leading source of insider news
and rumors on Apple Computer since
1997.</description>
        <language>en-us</language>
        <lastBuildDate>Fri, 10 Dec 2010
20:54:39 -0500</lastBuildDate>
        <image>
            <title>AppleInsider</title>
            <url>http://images.
appleinsider.com/toolbar_logo_text.gif</
url>
            <link>http://www.appleinsider.
com/</link>
```

FIGURE 8-10

How It Works

The `DownloadText()` method takes an URL of the text file to download and then returns the string of the text file downloaded. It basically opens an HTTP connection to the server and then uses an `InputStreamReader` object to read each character from the stream and save it in a `String` object.

Accessing Web Services

So far you have seen how to download images and text from the Web. The previous section showed how to download an RSS feed from a server. Very often, you need to download XML files and parse the contents (a good example of this is consuming Web services). Therefore, in this section you learn how to connect to a Web service using the HTTP `GET` method. Once the Web service returns a result in XML, you will extract the relevant parts and display its content using the `Toast` class.

For this example, the web method you will be using is from `http://services.aonaware.com/DictService/DictService.asmx?op=Define`. This web method is from a Dictionary Web service that returns the definitions of a given word.

The web method takes a request in the following format:

```
GET /DictService/DictService.asmx/Define?word=string HTTP/1.1
Host: services.aonaware.com
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
```

It returns a response in the following format:

```
<?xml version="1.0" encoding="utf-8"?>
<WordDefinition xmlns="http://services.aonaware.com/webservices/">
  <Word>string</Word>
  <Definitions>
    <Definition>
      <Word>string</Word>
      <Dictionary>
        <Id>string</Id>
        <Name>string</Name>
      </Dictionary>
      <WordDefinition>string</WordDefinition>
    </Definition>
    <Definition>
      <Word>string</Word>
      <Dictionary>
        <Id>string</Id>
        <Name>string</Name>
      </Dictionary>
      <WordDefinition>string</WordDefinition>
    </Definition>
  </Definitions>
</WordDefinition>
```

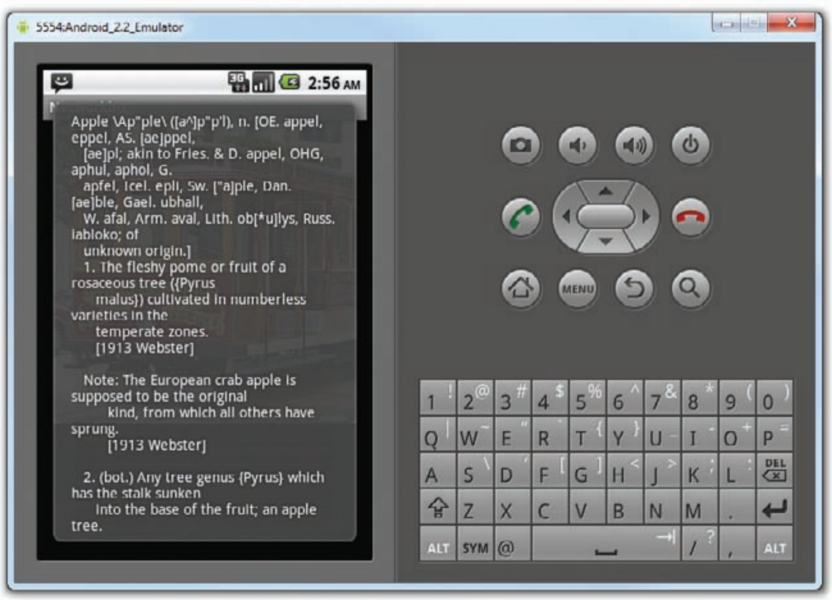
Hence, to obtain the definition of a word, you need to establish an HTTP connection to the web method and then parse the XML result that is returned. The following Try It Out shows you how.

TRY IT OUT Consuming Web Services

1. Using the same project created earlier, add the following statements in bold to the `MainActivity.java` file:

```
public class MainActivity extends Activity {
    ImageView img;

    private InputStream OpenHttpConnection(String urlString)
    throws IOException
    {
        //...
    }

    private Bitmap DownloadImage(String URL)
    {
        //...
    }

    private String DownloadText(String URL)
    {
        //...
    }

    private void WordDefinition(String word) {
        InputStream in = null;
        try {
            in = OpenHttpConnection(
"http://services.aonaware.com/DictService/DictService.asmx/Define?word=" + word);
            Document doc = null;
            DocumentBuilderFactory dbf =
                DocumentBuilderFactory.newInstance();
            DocumentBuilder db;
            try {
                db = dbf.newDocumentBuilder();
                doc = db.parse(in);
            } catch (ParserConfigurationException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            doc.getDocumentElement().normalize();

            //---retrieve all the <Definition> nodes---
            NodeList itemNodes =
                doc.getElementsByTagName("Definition");

            String strDefinition = "";
            for (int i = 0; i < definitionElements.getLength(); i++) {
                    Node itemNode = definitionElements.item(i);
                    if (itemNode.getNodeType() == Node.ELEMENT_NODE)
                    {
                        //---convert the Node into an Element---
                        Element definitionElement = (Element) itemNode;

                        //---get all the <WordDefinition> elements under
                        // the <Definition> element---
                        NodeList wordDefinitionElements =
                            (definitionElement).getElementsByTagName(
                            "WordDefinition");

                        strDefinition = "";
                        for (int j = 0; j < wordDefinitionElements.getLength(); j++) {
                            //---convert a <WordDefinition> Node into an Element---
                            Element wordDefinitionElement =
                                (Element) wordDefinitionElements.item(j);

                            //---get all the child nodes under the
                            // <WordDefinition> element---
                            NodeList textNodes =
                                ((Node) wordDefinitionElement).getChildNodes();

                            strDefinition +=
                                ((Node) textNodes.item(0)).getNodeValue() + ". ";
                        }

                        //---display the title---
                        Toast.makeText(getBaseContext(),strDefinition,
                            Toast.LENGTH_SHORT).show();
                    }
                }
            } catch (IOException e1) {
                Toast.makeText(this, e1.getLocalizedMessage(),
                    Toast.LENGTH_LONG).show();
                e1.printStackTrace();
            }
        }

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

            //---download an image---
            Bitmap bitmap =
                DownloadImage(
                "http://www.streetcar.org/mim/cable/images/cable-01.jpg");
            img = (ImageView) findViewById(R.id.img);
            img.setImageBitmap(bitmap);

            //---download an RSS feed---
            String str = DownloadText(
                "http://www.appleinsider.com/appleinsider.rss");
            Toast.makeText(getBaseContext(), str,
                    Toast.LENGTH_SHORT).show();

            //---access a Web service using GET---
            WordDefinition("Apple");
        }
    }
```

2. Press F11 to debug the application on the Android Emulator. Figure 8-11 shows the result of the Web service call being parsed and then displayed using the `Toast` class.



```
Apple \Ap"ple\ ([a^]p"p'l), n. [OE. appel,
eppel, AS. [ae]ppel,
   [ae]pl; akin to Fries. & D. appel, OHG,
aphul, aphol, G.
   apfel, Icel. epli, Sw. ["a]ple, Dan.
[ae]ble, Gael. ubhall,
   W. afal, Arm. aval, Lith. ob[*u]lys, Russ.
labloko; of
   unknown origin.]
   1. The fleshy pome or fruit of a
rosaceous tree ({Pyrus
      malus}) cultivated in numberless
varieties in the
      temperate zones.
      [1913 Webster]

   Note: The European crab apple is
supposed to be the original
         kind, from which all others have
sprung.
         [1913 Webster]

   2. (bot.) Any tree genus {Pyrus} which
has the stalk sunken
      into the base of the fruit; an apple
tree.
```

FIGURE 8-11

How It Works

The `WordDefinition()` method first opens an HTTP connection to the Web service, passing in the word that you are interested in:

```
                in = OpenHttpConnection(
    "http://services.aonaware.com/DictService/DictService.asmx/Define?word=" + word);
```

It then uses the `DocumentBuilderFactory` and `DocumentBuilder` objects to obtain a `Document` (DOM) object from an XML file (which is the XML result returned by the Web service):

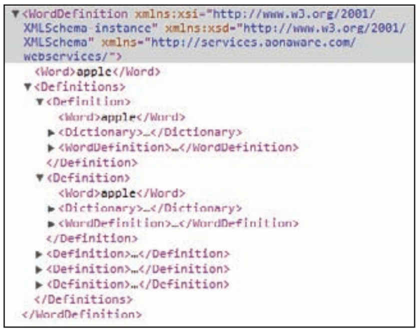
```
        Document doc = null;
        DocumentBuilderFactory dbf =
            DocumentBuilderFactory.newInstance();
        DocumentBuilder db;
        try {
```

```
        db = dbf.newDocumentBuilder();
        doc = db.parse(in);
    } catch (ParserConfigurationException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    doc.getDocumentElement().normalize();
```

Once the `Document` object is obtained, you will find all the elements with the `<Definition>` tag:

```
    //---retrieve all the <Definition> nodes---
    NodeList itemNodes =
        doc.getElementsByTagName("Definition");
```

Figure 8-12 shows the structure of the XML document returned by the Web service.



```
▼<WordDefinition xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/
  XMLSchema" xmlns="http://services.aonaware.com/
  webservices/">
   <Word>apple</Word>
  ▼<Definitions>
    ▼<Definition>
       <Word>apple</Word>
      ▶<Dictionary>…</Dictionary>
      ▶<WordDefinition>…</WordDefinition>
     </Definition>
    ▼<Definition>
       <Word>apple</Word>
      ▶<Dictionary>…</Dictionary>
      ▶<WordDefinition>…</WordDefinition>
     </Definition>
    ▶<Definition>…</Definition>
    ▶<Definition>…</Definition>
    ▶<Definition>…</Definition>
   </Definitions>
 </WordDefinition>
```

FIGURE 8-12

As the definition of a word is contained within the `<WordDefinition>` element, you then proceed to extract all the definitions:

```
    String strDefinition = "";
    for (int i = 0; i < definitionElements.getLength(); i++) {
        Node itemNode = definitionElements.item(i);
        if (itemNode.getNodeType() == Node.ELEMENT_NODE)
        {
            //---convert the Node into an Element---
            Element definitionElement = (Element) itemNode;

            //---get all the <WordDefinition> elements under
            // the <Definition> element---
            NodeList wordDefinitionElements =
                (definitionElement).getElementsByTagName(
                "WordDefinition");

            strDefinition = "";
            for (int j = 0; j < wordDefinitionElements.getLength(); j++) {
                //---convert a <WordDefinition> Node into an Element---
```

```
                            Element wordDefinitionElement =
                                (Element) wordDefinitionElements.item(j);

                            //---get all the child nodes under the
                            // <WordDefinition> element---
                            NodeList textNodes =
                                ((Node) wordDefinitionElement).getChildNodes();
                            //---get the first node, which contains the text---
                            strDefinition +=
                                ((Node) textNodes.item(0)).getNodeValue() + ". ";
                        }
                        //---display the title---
                        Toast.makeText(getBaseContext(),strDefinition,
                            Toast.LENGTH_SHORT).show();
                    }
                }
            } catch (IOException e1) {
                Toast.makeText(this, e1.getLocalizedMessage(),
                    Toast.LENGTH_LONG).show();
                e1.printStackTrace();
            }
```

The above loops through all the `<Definition>` elements and then for each `<Definition>` element it looks for a child element named `<WordDefinition>`. The text content of the `<WordDefinition>` element contains the definition of a word. The Toast class displays each word definition that is retrieved.

Performing Asynchronous Calls

So far, all the connections made in the previous few sections are all *synchronous* – that is, the connection to a server will not return until the data is received. In real life, this presents some problems due to network connections being inherently slow. When you connect to a server to download some data, the user interface of your application remains frozen until a response is obtained. In most cases, this is not acceptable. Hence, you need to ensure that the connection to the server is made in an asynchronous fashion.

The easiest way to connect to the server asynchronously is to use the `AsyncTask` class available in the Android SDK. Using `AsyncTask` enables you to perform background tasks in a separate thread and then return the result in a UI thread. Using this class enables you to perform background operations without needing to handle complex threading issues.

Using the previous example of downloading an image from the server and then displaying the image in an `ImageView`, you could wrap the code in an instance of the `AsyncTask` class, as shown below:

```
public class MainActivity extends Activity {
    ImageView img;

private class BackgroundTask extends AsyncTask
    <String, Void, Bitmap> {
        protected Bitmap doInBackground(String... url) {
            //---download an image---
            Bitmap bitmap = DownloadImage(url[0]);
            return bitmap;
        }

        protected void onPostExecute(Bitmap bitmap) {
            ImageView img = (ImageView) findViewById(R.id.img);
            img.setImageBitmap(bitmap);
        }
    }

    private InputStream OpenHttpConnection(String urlString)
    throws IOException
    {
        ...
    }
```

Basically, you defined a class that extends the `AsyncTask` class. In this case, there are two methods within the `BackgroundTask` class — `doInBackground()` and `onPostExecute()`. You put all the code that needs to be run asynchronously in the `doInBackground()` method. When the task is completed, the result is passed back via the `onPostExecute()` method. The `onPostExecute()` method is executed on the UI thread, hence it is thread safe to update the `ImageView` with the bitmap downloaded from the server.

***NOTE** You will learn more about the `AsyncTask` class in Chapter 10 which covers developing services in Android.*

To perform the asynchronous tasks, simply create an instance of the `BackgroundTask` class and call its `execute()` method:

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
            new BackgroundTask().execute(
                "http://www.streetcar.org/mim/cable/images/cable-01.jpg");
}
```

SUMMARY

This chapter described the various ways to communicate with the outside world. You first learned how to send and receive SMS messages. You then learned how to send e-mail messages from within your Android application. Besides SMS and e-mail, another way to communicate with the outside world is through the use of the HTTP protocol. Using the HTTP protocol, you can download data from a web server. One good application of this is to talk to Web services, whereby you need to parse XML files.

EXERCISES

1. Name the two ways in which you can send SMS messages in your Android application.
2. Name the permissions you need to declare in your `AndroidManifest.xml` file for sending and receiving SMS messages.
3. How do you notify an activity from a `BroadcastReceiver`?
4. Name the permissions you need to declare in your `AndroidManifest.xml` file for an HTTP connection.

Answers to Exercises can be found in Appendix C.

▶ WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
Programmatically sending SMS messages	Use the `SmsManager` class.
Getting feedback on messages sent	Use two `PendingIntent` objects in the `sendTextMessage()` method.
Sending SMS messages using Intent	Set the intent type to "`vnd.android-dir/mms-sms`".
Receiving SMS messages	Implement a `BroadcastReceiver` and set it in the `AndroidManifest.xml` file.
Sending e-mail using Intent	Set the intent type to "`message/rfc822`".
Establishing an HTTP connection	Use the `HttpURLConnection` class.
Accessing Web services	Use the `Document`, `DocumentBuilderFactory`, and `DocumentBuilder` classes to parse the XML result returned by the Web service.