10

Rad u mrežnom okruženju

Aplikacije za rad u mrežnom okruženju vrednije su korisniku jer njihov sadržaj može biti dinamičan i interaktivan. Mrežno okruženje podrazumeva razne mogućnosti – od društevnih mreža (engl. *social networking*) do rada u oblaku (engl. *cloud computing*).

U ovom poglavlju obrađene su sledeće teme: stanje mreže, SMS poruke, aplikacije koje koriste internet resurse i aplikacije za upotrebu društvenih mreža. Poznavanje stanja mreže je važno za aplikacije koje pribavljaju ili ažuriraju informacije dostupne preko mrežne veze. SMS je komponenta za komunikacioni servis koja omogućava razmenu kratkih tekstualnih poruka između mobilnih telefona. Aplikacije koje koriste internet resurse rade s veb sadržajem kao što su HTML (HyperText Markup Language), XML (eXtensible Markup Language) i JSON (JavaScript Object Notation). Aplikacije za upotrebu društvenih mreža, kao što je Twitter, važni su načini uspostavljanja veza između ljudi.

Reagovanje na promene stanja u mreži

Pribavljanje informacija o tome da li je uređaj povezan s određenom mrežom i kako, vrlo su važan deo procesa razvoja aplikacija za Android. U aplikacijama koje primaju podatke u obliku tokova s mrežnog servera, možda treba upozoriti korisnika da velika količina preuzetih podataka može biti naplaćena s korisnikovog naloga. Problemi u vezi s vremenom odziva aplikacija takođe mogu biti razlog za zabrinutost. Izvršavanje nekoliko jednostavnih upita omogućava korisnicima da utvrde da li je njihov uređaj povezan u mrežu i kako da reaguju kada se stanje veze promeni.

Recept: Utvrđivanje da li je moguće uspostavljanje veze s mrežom

Klasa `ConnectivityManager` omogućava da utvrdite da je moguće uspostavljanje veze između uređaja i određenog udaljenog servisa. Ovaj recept možete iskoristiti da biste utvrdili koji su mrežni interfejsi na uređaju povezani s datom mrežom. U listingu 10.1, pomoću objekta tipa `ConnectivityManager` prikazuje se da li je uređaj povezan u mrežu preko Wi-Fi ili Bluetooth veze.

Listing 10.1 src/com/cookbook/connectivitycheck/MainActivity.java

```
package com.cookbook.connectivitycheck;

import android.app.Activity;
import android.content.Context;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends Activity {
   TextView tv;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      tv = (TextView) findViewById(R.id.tv_main);
      try {
         String service = Context.CONNECTIVITY_SERVICE;
         ConnectivityManager cm = (ConnectivityManager)
              getSystemService(service);
         NetworkInfo activeNetwork = cm.getActiveNetworkInfo();

         boolean isWiFi = activeNetwork.getType() == ConnectivityManager.
              TYPE_WIFI;
         boolean isBT = activeNetwork.getType() == ConnectivityManager.
              TYPE_BLUETOOTH;

         tv.setText("WiFi connected: "+isWiFi+"\nBluetooth connected: "+isBT);
      } catch(Exception nullPointerException)  {
         tv.setText("No connected networks found");
      }
   }
}
```

U listingu 10.1, pomoću konstanti `TYPE_WIFI` i `TYPE_BLUETOOTH` ispituje se povezanost s odgovarjućom vrstom mreže. Osim konstanti `TYPE_WIFI` i `TYPE_BLUETOOTH`, povezanost se može utvrđivati i pomoću sledećih konstanti:

- `TYPE_DUMMY` – Za simulirane veze za prenošenje podataka
- `TYPE_ETHERNET` – Za podrazumevanu Ethernet vezu
- `TYPE_MOBILE` – Za podrazumevanu vezu s mobilnom mrežom
- `TYPE_MOBILE_DUN` – Za vezu s drugim uređajem uspostavljenu pomoću metode DUN (dial-up networking) radi prenošenja podataka mobilnom mrežom

- `TYPE_MOBILE_HIPRI` – Za prenošenje podataka mobilnom mrežom i visokim prioritetom
- `TYPE_MOBILE_MMS` – Za prenošenje MMS poruka mobilnom mrežom
- `TYPE_MOBILE_SUPL` – Za prenošenje podataka mobilnom mrežom po standardu SUPL
- `TYPE_WIMAX` – Za podrazumevanu vezu tipa WiMAX

Slika 10.1 prikazuje primer aplikacije u kojoj radi kôd iz listinga 10.1. Uprkos tome što je dozvoljeno uspostavljanje Bluetooth veza, prikazuje se rezultat `false` zato što aplikacija nije uspostavila nijednu aktivnu vezu tim putem.

Slika 10.1 Utvrđivanje da li je uređaj umrežen

Recept: Otkrivanje promena povezanosti uređaja s mrežom

Ako aplikacija treba da reaguje na promene povezanosti uređaja s mrežom, za otkrivanje tih promena može se iskoristiti prijemnik sistemskih poruka (objekat klase `BroadcastReceiver`).

Prijemnik sistemskih poruka može se deklarisati u manifestu aplikacije ili on može biti potklasa deklarisana unutar klase glavne aktivnosti. U ovom receptu koristi se potklasa, a u metodama `onCreate()` i `onDestroy()` prijemnik se registruje, odnosno deregistruje.

Pošto ovaj recept ispituje povezanost uređaja s mrežom, u manifest aplikacije treba dodati sledeća ovlašćenja:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Listing 10.2 prikazuje kôd koji utvrđuje promene povezanosti uređaja s mrežom. Kada otkrije promenu, aplikacija prikazuje kratku poruku na ekranu kojom informiše korisnika o nastaloj promeni.

Listing 10.2 src/com/cookbook/connectivitychange/MainActivity.java

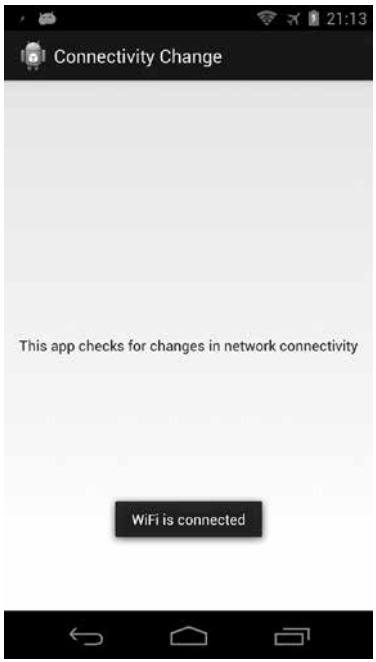
```
package com.cookbook.connectivitychange;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.Bundle;
import android.widget.Toast;

public class MainActivity extends Activity {

   private ConnectivityReceiver receiver = new ConnectivityReceiver();

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      IntentFilter filter = new IntentFilter(ConnectivityManager.CONNECTIVITY_
           ACTION);
         receiver = new ConnectivityReceiver();
         this.registerReceiver(receiver, filter);
   }

   @Override
      public void onDestroy() {
         super.onDestroy();
         if (receiver != null) {
```

```
            this.unregisterReceiver(receiver);
         }
      }

   public class ConnectivityReceiver extends BroadcastReceiver {

   @Override
   public void onReceive(Context context, Intent intent) {
      ConnectivityManager conn =
(ConnectivityManager)context.getSystemService(Context.CONNECTIVITY_SERVICE);
      NetworkInfo networkInfo = conn.getActiveNetworkInfo();

      if (networkInfo != null && networkInfo.getType() == ConnectivityManager.
➥TYPE_WIFI) {
         Toast.makeText(context, "WiFi is connected", Toast.LENGTH_SHORT).
            ➥show();
      } else if (networkInfo != null) {
         Toast.makeText(context, "WiFi is disconnected", Toast.LENGTH_SHORT).
            ➥show();
      } else {
         Toast.makeText(context, "No active connection", Toast.LENGTH_SHORT).
            ➥show();
      }
      }
   }
}
```

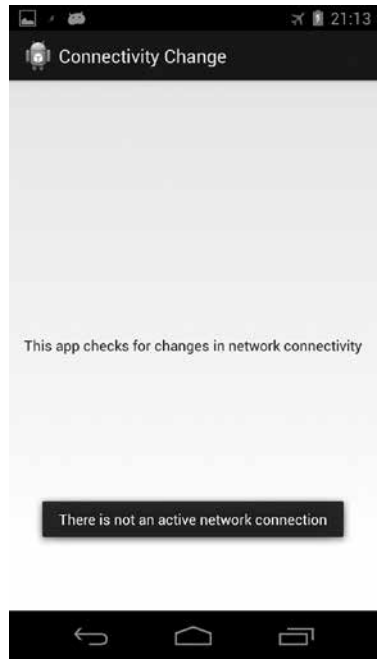
Slika 10.2 prikazuje poruku koja se prikazuje kada je uspostavljena Wi-Fi veza. Slika 10.3 prikazuje poruku koja se pojavljuje kada nije uključeno uspostavljanje ni Wi-Fi veza ni veza s mobilnom mrežom.

SMS poruke

Androidovo radno okruženje omogućava upotrebu ceolokupne SMS funkcionalnosti pomoću klase `SmsManager`. U starijim verzijama Androida, klasa `SmsManager` nalazila se u paketu `android.telephony.gsm`. Od Androida 1.5 nadalje, gde klasa `SmsManager` podržava i GSM i CDMA standarde za mobilnu telefoniju, ta klasa je smeštena u paket `android.telephony`.



Slika 10.2 Kada je uključeno uspostavljanje Wi-Fi veza, na ekranu se prikazuje poruka koja informiše korisnika o uspostavljanju veze



Slika 10.3 Pošto uspostavljanje Wi-Fi veza i prenošenje podataka mobilnom mrežom nije uključeno, poruka na ekranu informiše korisnika da njegov uređaj nije umrežen ni na koji način

Slanje SMS poruke pomoću klase `SmsManager` vrlo je jednostavno:

1. U datoteci **AndroidManifest.xml** zadajte ovlašćenje za slanje SMS poruka:
   ```
   <uses-permission android:name="android.permission.SEND_SMS">
   ```
1. Pozovite statičku metodu `SmsManager.getDefault()` da biste dobili instancu klase `SmsManager`:
   ```
   SmsManager mySMS = SmsManager.getDefault();
   ```
2. Zadajte telefonski broj primaoca poruke i tekst poruke koju šaljete. Pozovite metodu `sendTextMesssage()` da biste poslali poruku drugom uređaju:
   ```
   String destination = "0631234567";
   String msg = "Šaljem svoju prvu poruku";
   mySMS.sendTextMessage(destination, null, msg, null, null);
   ```

To je dovoljno za slanje SMS poruke. Međutim, tri parametra koja u prethodnom pozivu imaju vrednost `null` mogu se iskoristiti na sledeći način:

- Drugi parametar je broj SMS servisnog centra preko kojeg se šalje proruka. Zadajte vrednost `null` da biste koristili podrazumevani servisni centar vašeg davaoca usluga (operatera).

- Četvrti parametar je objekat tipa `PendingIntent` pomoću kojeg aplikacija prati da li je SMS poruka uspešno poslata.
- Peti parametar je objekat tipa `PendingIntent` pomoću kojeg aplikacija prati da li je SMS poruka primljena.

Da biste iskoristili četvrti i peti parametar, morate deklarisati nameru za poslatu poruku i nameru za primljenu poruku:

```
String SENT_SMS_FLAG = "SENT_SMS";
String DELIVER_SMS_FLAG = "DELIVER_SMS";

Intent sentIn = new Intent(SENT_SMS_FLAG);
PendingIntent sentPIn = PendingIntent.getBroadcast(this,0,sentIn,0);

Intent deliverIn = new Intent(SENT_SMS_FLAG);
PendingIntent deliverPIn
          = PendingIntent.getBroadcast(this,0,deliverIn,0);
```

Da biste zatim primili rezultat, morate registrovati klasu `BroadcastReceiver` za svaki objekat `PendingIntent`:

```
BroadcastReceiver sentReceiver = new BroadcastReceiver(){
    @Override public void onReceive(Context c, Intent in) {
       switch(getResultCode()){
          case Activity.RESULT_OK:
             //SMS poruka je uspešno poslata;
             break;
          default:
             //slanje SMS poruke nije uspelo
             break;
       }
    }
};

BroadcastReceiver deliverReceiver = new BroadcastReceiver(){
    @Override public void onReceive(Context c, Intent in) {
       //akcije nakon prijema SMS poruke
    }
};

registerReceiver(sentReceiver, new IntentFilter(SENT_SMS_FLAG));
registerReceiver(deliverReceiver, new IntentFilter(DELIVER_SMS_FLAG));
```

Dužina većine SMS poruka ograničena je na 140 znakova po poruci. Da bi tekst poruke uvek bio unutar te granice, pozovite metodu `divideMessage()` koja deli tekst na blokove maksimalne dozvoljene veličine. Zatim pozovite metodu `sendMultipartTextMessage()` umesto metode `sendTextMessage()`. Jedina razlika je upotreba liste tipa `ArrayList` za poruke i namere koje nisu izvršene:

```
ArrayList<String> multiSMS = mySMS.divideMessage(msg);
ArrayList<PendingIntent> sentIns = new ArrayList<PendingIntent>();
ArrayList<PendingIntent> deliverIns = new ArrayList<PendingIntent>();

for(int i=0; i< multiSMS.size(); i++){
   sentIns.add(sentIn);
   deliverIns.add(deliverIn);
}
mySMS.sendMultipartTextMessage(destination, null,
                               multiSMS, sentIns, deliverIns);
```

Recept: Automatsko slanje SMS poruke kao odziv na prijem SMS poruke

Pošto većinu SMS poruka primalac pročita možda i više sati kasnije, u ovom receptu se automatski šalje SMS poruka kao odziv na prijem SMS poruke. To obavlja Androidov servis koji radi u pozadini i obrađuje dolazne SMS poruke. Drugo rešenje je da u datoteci **AndroidManifest.xml** registrujete prijemnik sistemskih poruka.

Aplikacija mora da deklariše ovlašćenje za slanje i prijem SMS poruka u datoteci **AndroidManifest.xml**, kao što je prikazano u listingu 10.3. U tom listingu je deklarisana i glavna aktivnost, `SMSResponder`, koja formira automatsku odzivnu poruku, kao i servis `ResponderService`, koji šalje odgovor nakon prijema SMS poruke.

Listing 10.3 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.cookbook.SMSResponder"
      android:versionCode="1"
      android:versionName="1.0">
   <application android:icon="@drawable/icon"
                android:label="@string/app_name">
      <activity android:name=".SMSResponder"
                android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <service android:enabled="true" android:name=".ResponderService">
      </service>
   </application>

   <uses-permission android:name="android.permission.RECEIVE_SMS"/>
   <uses-permission android:name="android.permission.SEND_SMS"/>
</manifest>
```

Listing 10.4 prikazuje sadržaj glavnog ekrana aplikacije u kojem element `LinearLayout` sadrži tri prikaza: polje tipa `TextView` koje prikazuje tekst automatske odzivne poruke, dugme kojim korisnik potvrđuje izmene koje je unutar aplikacije uneo u tekst odzivne poruke i polje tipa `EditText` gde korisnik može da upiše tekst odzivne poruke.

Listing 10.4 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
      <TextView android:id="@+id/display"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/hello"
      android:textSize="18dp"
      />
      <Button android:id="@+id/submit"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Change my response"
      />
      <EditText
      android:id="@+id/editText"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      />
</LinearLayout>
```

Listing 10.5 prikazuje glavnu aktivnost; ona pokreće servis koji osluškuje prijem SMS poruka i automatski šalje odgovore na njih. Ta aktivnost takođe omogućava korisniku da izmeni tekst odzivne poruke i da ga snimi u objekat tipa `SharedPreference` za buduću upotrebu.

Listing 10.5 src/com/cookbook/SMSresponder/SMSResponder.java

```
package com.cookbook.SMSresponder;

import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.util.Log;
import android.view.View;
```

```
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class SMSResponder extends Activity {
   TextView tv1;
   EditText ed1;
   Button bt1;
   SharedPreferences myprefs;
   Editor updater;
   String reply=null;

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

      myprefs = PreferenceManager.getDefaultSharedPreferences(this);
      tv1 = (TextView) this.findViewById(R.id.display);
      ed1 = (EditText) this.findViewById(R.id.editText);
      bt1 = (Button) this.findViewById(R.id.submit);

      reply = myprefs.getString("reply",
              "Thank you for your message. I am busy now. "
              + "I will call you later");
      tv1.setText(reply);

      updater = myprefs.edit();
      ed1.setHint(reply);
      bt1.setOnClickListener(new OnClickListener() {
         public void onClick(View view) {
            updater.putString("reply", ed1.getText().toString());
            updater.commit();
            SMSResponder.this.finish();
         }
      });

      try {
         // Pokretnje servisa
         Intent svc = new Intent(this, ResponderService.class);
         startService(svc);
      }
      catch (Exception e) {
         Log.e("onCreate", "service creation problem", e);
      }
   }
}
```

Veći deo koda pripada servisu, kao što se vidi u listingu 10.6. Servis prvo učitava objekat `SharedPreferences` za aplikaciju, a zatim registruje prijemnik sistemskih poruka koji osluškuje primanje i slanje SMS poruka. U ovom primeru se ne koristi prijemnik sistemskih poruka za odlazne SMS poruke, ali je ipak prikazan da bi kôd bio kompletan.

Prijemnik sistemskih poruka koji obrađuje dolazne SMS poruke učitava PDU (*protocol description unit* – jedinica za opisivanje protokola) u obliku objekta tipa `Bundle`, koji sadrži tekst SMS poruke i dodatne SMS metapodatke, koje raspodeljuje u niz tipa `Object`. Metoda `createFromPdu()` pretvara niz tipa `Object` u objekat tipa `SmsMessage`. Zatim se pomoću metode `getOriginatingAddress()` poruci može dodati telefonski broj pošiljaoca, a metoda `getMessageBody()` vraća tekst poruke.

U ovom receptu, pošto servis učita adresu pošiljaoca, poziva metodu `respond()`. Ta metoda pokušava da učita podatke koji se čuvaju u objektu `SharedPreferences` da bi formirala automatsku odzivnu poruku. Ako nema takvih podataka, koristi podrazumevanu poruku. Zatim pravi dva objekta tipa `PendingIntent` koji predstavljaju status slanja i status prijema poruke. Metoda `divideMessage()` obezbeđuje da poruka ne premaši dozvoljenu dužinu. Posle pripreme svih podataka, servis ih šalje pomoću metode `sendMuiltTextMessage()`.

Listing 10.6 src/com/cookbook/SMSresponder/ResponderService.java

```
package com.cookbook.SMSresponder;

import java.util.ArrayList;
import android.app.Activity;
import android.app.PendingIntent;
import android.app.Service;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.os.IBinder;
import android.preference.PreferenceManager;
import android.telephony.SmsManager;
import android.telephony.SmsMessage;
import android.util.Log;
import android.widget.Toast;

public class ResponderService extends Service {
   //Akcija koju pokreće OS Android nakon prijema SMS poruke.
   private static final String RECEIVED_ACTION =
            "android.provider.Telephony.SMS_RECEIVED";
   private static final String SENT_ACTION="SENT_SMS";
```

```
private static final String DELIVERED_ACTION="DELIVERED_SMS";

String requester; String reply="";
SharedPreferences myprefs;

@Override
public void onCreate() {
   super.onCreate();
   myprefs = PreferenceManager.getDefaultSharedPreferences(this);

   registerReceiver(sentReceiver, new IntentFilter(SENT_ACTION));
   registerReceiver(deliverReceiver, new IntentFilter(DELIVERED_ACTION));

   IntentFilter filter = new IntentFilter(RECEIVED_ACTION);
   registerReceiver(receiver, filter);

   IntentFilter attemptedfilter = new IntentFilter(SENT_ACTION);
   registerReceiver(sender,attemptedfilter);
}

private BroadcastReceiver sender = new BroadcastReceiver(){
   @Override
   public void onReceive(Context c, Intent i) {
      if(i.getAction().equals(SENT_ACTION)) {
         if(getResultCode() != Activity.RESULT_OK) {
            String recipient = i.getStringExtra("recipient");
            requestReceived(recipient);
         }
      }
   }
};

BroadcastReceiver sentReceiver = new BroadcastReceiver() {
   @Override
   public void onReceive(Context c, Intent in) {
      switch(getResultCode()) {
         case Activity.RESULT_OK:
            // SMS poruka je uspešno poslata;
            smsSent();
            break;
         default:
            //slanje SMS poruke nije uspelo
            smsFailed();
            break;
      }
   }
};
```

```
public void smsSent() {
   Toast.makeText(this, "SMS sent", Toast.LENGTH_SHORT);
}

public void smsFailed() {
   Toast.makeText(this, "SMS sent failed", Toast.LENGTH_SHORT);
}

public void smsDelivered() {
   Toast.makeText(this, "SMS delivered", Toast.LENGTH_SHORT);
}

BroadcastReceiver deliverReceiver = new BroadcastReceiver() {
   @Override public void onReceive(Context c, Intent in) {
      //SMS poruka je primljena
      smsDelivered();
   }
};

public void requestReceived(String f) {
   Log.v("ResponderService","In requestReceived");
   requester=f;
}

BroadcastReceiver receiver = new BroadcastReceiver() {
   @Override
   public void onReceive(Context c, Intent in) {
      Log.v("ResponderService","On Receive");
      reply="";
      if(in.getAction().equals(RECEIVED_ACTION)) {
         Log.v("ResponderService","On SMS RECEIVE");

         Bundle bundle = in.getExtras();
         if(bundle!=null) {
            Object[] pdus = (Object[])bundle.get("pdus");
            SmsMessage[] messages = new SmsMessage[pdus.length];
            for(int i = 0; i<pdus.length; i++) {
               Log.v("ResponderService","FOUND MESSAGE");
               messages[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
            }
            for(SmsMessage message: messages) {
               requestReceived(message.getOriginatingAddress());
            }
            respond();
         }
      }
   }
};
```

```
    @Override
    public void onStart(Intent intent, int startId) {
       super.onStart(intent, startId);
    }

    public void respond() {
       Log.v("ResponderService","Responding to " + requester);
          reply = myprefs.getString("reply",
                  "Thank you for your message. I am busy now. "
                  "I will call you later");
       SmsManager sms = SmsManager.getDefault();
       Intent sentIn = new Intent(SENT_ACTION);
       PendingIntent sentPIn = PendingIntent.getBroadcast(this,0,sentIn,0);
       Intent deliverIn = new Intent(DELIVERED_ACTION);
       PendingIntent deliverPIn = PendingIntent.getBroadcast(this,0,deliver
            In,0);
       ArrayList<String> Msgs = sms.divideMessage(reply);
       ArrayList<PendingIntent> sentIns = new ArrayList<PendingIntent>();
       ArrayList<PendingIntent> deliverIns = new ArrayList<PendingIntent>();

       for(int i=0; i< Msgs.size(); i++) {
          sentIns.add(sentPIn);
          deliverIns.add(deliverPIn);
       }

       sms.sendMultipartTextMessage(requester, null, Msgs, sentIns, deliverIns);
    }

    @Override
    public void onDestroy() {
       super.onDestroy();
       unregisterReceiver(receiver);
       unregisterReceiver(sender);
    }

    @Override public IBinder onBind(Intent arg0) {
       return null;
    }
}
```

Rad s veb sadržajem

Da biste pokrenuli čitač veba (engl. *web browser*) radi prikazivanja veb sadržaja, možete iskoristiti implicitnu nameru `ACTION_VIEW`, kao što je opisano u poglavlju 2, „Osnove aplikacija: aktivnosti i namere“. Na primer:

```
Intent i = new Intent(Intent.ACTION_VIEW);
i.setData(Uri.parse("http://www.google.com"));
startActivity(i);
```

Programer može i sam da napravi čitač veba, pomoću prikaza tipa `WebView`, što je objekat tipa `View` koji prikazuje veb sadržaj. Kao i svaki drugi prikaz, može zauzeti ceo ekran ili samo deo ekrana aktivnosti. Klasa `WebView` vizuelizuje veb stranice pomoću WebKita, što je mašina za čitač veba otvorenog koda koja se koristi u Appleovom čitaču Safari.

Recept: Izrada vlastitog čitača veba

Objekat tipa `WebView` možete napraviti na dva načina. Možete ga instancirati pomoću njegovog konstruktora:

```
WebView webview = new WebView(this);
```

Druga mogućnost je da objekat tipa `WebView` definišete u datoteci sadržaja ekrana i deklarišete u kodu aktivnosti:

```
WebView webView = (WebView) findViewById(R.id.webview);
```

Pošto formirate taj objekat, možete prikazati sadržaj određene stranice tako što ćete pozvati njegovu metodu `loadURL()`:

```
webview.loadUrl("http://www.google.com/");
```

Klasa `WebSettings` omogućava da zadate osobine čitača veba. Na primer, pomoću metode `setBlockNetworkImage()` možete blokirati prikazivanje slika sa veba da biste smanjili količinu podataka koja se prenosi. Veličinu fonta kojim se prikazuje veb sadržaj možete zadati pomoću metode `setDefaultFontSize()`. U primeru je prikazano podešavanje i drugih uobičajenih parametara:

```
WebSettings webSettings = webView.getSettings();
webSettings.setSaveFormData(false);
webSettings.setJavaScriptEnabled(true);
webSettings.setSavePassword(false);
webSettings.setSupportZoom(true);
```

Recept: Upotreba HTTP metode GET

Osim pokretanja čitača veba ili upotrebe klase `WebView` radi ugradnje u aktivnost čitača zasnovanog na WebKitu, projektanti mogu da prave i matične internet aplikacije. To znači da aplikacija radi sa sirovim podacima koji se nalaze na internetu, kao što su slike, multimedijske datoteke i XML podaci. Aplikacija može direktno učitavati podatke s kojima radi. To je važno za izradu aplikacija za upotrebu društvenih mreža. Komuniciranje u umreženom okruženju omogućavaju paketi `java.net` i `android.net`.

U ovom receptu, pomoću metode GET protokola HTTP učitavaju se XML ili JSON podaci (videti pregled na www.json.org/). Recept ilustruje upotrebu

Googleovog API-ja za pretraživanje REST (Representational State Transfer), a koristi se sledeći upit:

```
http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=
```

Da biste pretraživali po određenom pojmu, samo ga dodajte na kraj upita. Na primer, kada pretražujete po pojmu NBA (National Basketball Association), sledeći upit vraća JSON podatke:

```
http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=NBA
```

Da bi se ta aktivnost izvršavala, potrebno joj je ovlašćenje za pristup internetu, pa zato treba dodati sledeći red u datoteku **AndroidManifest.xml**:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Sadržaj glavnog ekrana prikazan je u listingu 10.7. Ekran sadrži tri prikaza: `EditText`, u koji korisnik upisuje traženi pojam, `Button` koji pokreće pretraživanje i `TextView` koji prikazuje rezultate pretrage.

Listing 10.7 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   >
      <EditText
      android:id="@+id/editText"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:singleLine="true"
      />
      <Button
      android:id="@+id/submit"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Search"
      />
      <TextView
      android:id="@+id/display"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:text="@string/hello"
      android:textSize="18dp"
      />
</LinearLayout>
```

Listing 10.8 prikazuje glavnu aktivnost. Prvo se u metodi `onCreate()` inicijalizuju tri elementa na ekranu. U kodu klase `OnClickListener` koja je pridružena dugmetu, poziva se funkcija `SearchRequest()`, koja sastavlja URL upita u formatu Google REST API a zatim inicijalizuje instancu klase URL i pomoću nje pravi instancu klase `HttpURLConnection`.

Instanca `HttpURLConnection` učitava status veze sa HTTP serverom. Kada objekat `HttpURLConnection` vrati rezultat čiji je kôd `HTTP_OK`, to znači da je cela HTTP transakcija uspešno obavljena. Zatim se JSON podaci koje je vratila HTTP transakcija prenose u jedan znakovni niz. To se radi tako što se objekat tipa `InputStreamReader` prosledi objektu tipa `BufferReader` koji učitava podatke i formira instancu tipa `String`. Nakon učitavanja HTTP rezultata, poziva se funkcija `ProcessResponse()` koja podatke iz JSON formata prevodi u znakovne nizove.

Listing 10.8 src/com/cookbook/internet/search/GoogleSearch.java

```
package com.cookbook.internet.search;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.security.NoSuchAlgorithmException;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class GoogleSearch extends Activity {
   /** Poziva se pri generisanju aktivnosti. */
   TextView tv1;
   EditText ed1;
   Button bt1;
   static String url =
"http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=";
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);
   tv1 = (TextView) this.findViewById(R.id.display);
   ed1 = (EditText) this.findViewById(R.id.editText);
   bt1 = (Button) this.findViewById(R.id.submit);

   bt1.setOnClickListener(new OnClickListener() {
      public void onClick(View view) {
         if(ed1.getText().toString()!=null) {
            try{
               processResponse(searchRequest(ed1.getText().toString()));
            } catch(Exception e) {
               Log.v("Exception Google search",
                     "Exception:"+e.getMessage());
            }
         }
         ed1.setText("");
      }
   });
}

public String searchRequest(String searchString)
                  throws MalformedURLException, IOException {
   String newFeed=url+searchString;
   StringBuilder response = new StringBuilder();
   Log.v("gsearch","gsearch url:"+newFeed);
   URL url = new URL(newFeed);

   HttpURLConnection httpconn = (HttpURLConnection) url.openConnection();

   if(httpconn.getResponseCode()==HttpURLConnection.HTTP_OK) {
      BufferedReader input = new BufferedReader(
         new InputStreamReader(httpconn.getInputStream()), 8192);
      String strLine = null;
      while ((strLine = input.readLine()) != null) {
         response.append(strLine);
      }
      input.close();
   }
   return response.toString();
}

public void processResponse(String resp) throws IllegalStateException,
               IOException, JSONException, NoSuchAlgorithmException {
   StringBuilder sb = new StringBuilder();
   Log.v("gsearch","gsearch result:"+resp);
```

```
        JSONObject mResponseObject = new JSONObject(resp);
        JSONObject responObject = mResponseObject.getJSONObject("responseData");
        JSONArray array = responObject.getJSONArray("results");
        Log.v("gsearch","number of resultst:"+array.length());
        for(int i = 0; i<array.length(); i++) {
           Log.v("result",i+"] "+array.get(i).toString());
           String title = array.getJSONObject(i).getString("title");
           String urllink = array.getJSONObject(i).getString("visibleUrl");
           sb.append(title);
           sb.append("\n");
           sb.append(urllink);
           sb.append("\n");
        }
        tv1.setText(sb.toString());
    }
}
```

Za detaljno razumevanje celog mehanizma potrebno je poznavanje strukture JSON podataka. U ovom primeru, Googleov REST API smešta sve primljene podatke u niz `JSONArray`. Slika 10.4 prikazuje ekran s rezultatima pretrage po pojmu NBA.

Slika 10.4 Rezultati pretraživanja pomoću API-ja Google REST

Imajte u vidu da će ovaj recept raditi samo na Android projektima koji koriste API Level stariji od 11. Razlog je to što se mrežni zahtevi izvršavaju u glavnoj niti. U sledećem receptu, „Upotreba HTTP metode post“, koristi se `AsyncTask` za ispravljanje generisane greške `NetworkOnMainThreadException`.

Recept: Upotreba HTTP metode POST

Ponekad je potrebno učitati sa interneta sirove binarne podatke, kao što je datoteka koja sadrži sliku, video ili audio zapis. To se može uraditi pomoću metode POST protokola HTTP tako što se pozove metoda `setRequestMethod()`, na primer:

```
httpconn.setRequestMethod(POST);
```

Pristupanje podacima koji se nalaze na internetu može potrajati a rezultati nisu uvek predvidljivi. Iz tih razloga, kad god su potrebni podaci iz mreže, treba pokrenuti zasebnu nit.

Osim metoda koje su prikazane u poglavlju 3, „Niti, servisi, prijemnici i alarmi“, postoji i ugrađena Androidova klasa `AsyncTask` koja omogućava izvršavanje operacija u pozadini i prosleđuje njihove rezultate u nit UI, pri čemu ne morate sami manipulisati nitima ili procedurama za obradu događaja. To omogućava asinhronu upotrebu metode POST, pomoću sledećeg koda :

```
private class MyGoogleSearch extends AsyncTask<String, Integer, String> {

    protected String doInBackground(String... searchKey) {

        String key = searchKey[0];

        try {
            return searchRequest(key);
        } catch(Exception e) {
            Log.v("Exception Google search",
                "Exception:"+e.getMessage());
            return "";
        }
    }

    protected void onPostExecute(String result) {
        try {
            processResponse(result);
        } catch(Exception e) {
            Log.v("Exception Google search",
                "Exception:"+e.getMessage());
        }
    }
}
```

Ovaj blok koda možete dodati na kraj aktivnosti **GoogleSearch.java** (listing 10.8). Rezultat aktivosti je isti, uz dodatnu izmenu koda za osluškivač `OnClickListener` koji je pridružen dugmetu:

```
new MyGoogleSearch().execute(ed1.getText().toString());
```

Recept: Upotreba prikaza tipa WebView

Prikazi tipa WebView korisni su kada treba prikazivati sadržaj koji se može menjati „poluredovno“ ili za podatke koji se mogu menjati a da pritom nije potrebno ažurirati i samu aplikaciju. Objekat tipa WebView takođe se može iskoristiti kada veb aplikacija treba da pristupa nekim klijentskim mogućnostima operativnog sistema Android, kao što je upotreba sistema za prikazivanje kratkih poruka na ekranu.

Da biste aplikaciji dodali prikaz WebView, u XML datoteku definicije ekrana dodajte sledeće:

```
<WebView  xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/webview"
  android:layout_width="match_parent"
  android:layout_height="match_parent" />
```

U manifest aplikacije treba dodati sledeće ovlašćenje:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Da biste napravili jednostavnu stranicu koja ne omogućava interakciju s korisnikom, dodajte sledeći blok koda u metodu `onCreate()` glavne aktivnosti:

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.loadUrl("http://www.example.com/");
```

Da biste na stranici koju prikazuje objekat WebView omogućili i rad JavaScript koda, treba izmeniti podešavanja tog objekta, što se radi pomoću objekta `WebSettings`. Postupak je sledeći:

```
WebSettings webSettings = myWebView.getSettings();
webSettings.setJavaScriptEnabled(true);
```

Da biste izvršavali metode u JavaScript kodu, morate definisati klasu koju ćete koristiti kao interfejs prema JavaScript kodu. Listing 10.9 prikazuje primer aktivnosti koja sadrži sve potrebne elemente za tu namenu.

Listing 10.9 src/com/cookbook/viewtoaweb/MainActivity.java

```
package com.cookbook.viewtoaweb;
import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.webkit.JavascriptInterface;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.widget.Toast;

public class MainActivity extends Activity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        WebView myWebView = (WebView) findViewById(R.id.webview);
        WebSettings webSettings = myWebView.getSettings();
        webSettings.setJavaScriptEnabled(true);
        myWebView.addJavascriptInterface(new WebAppInterface(this), "Android");
        myWebView.loadUrl("http://www.devcannon.com/androidcookbook/chapter10/
             webview/");
    }

    public class WebAppInterface {
        Context context;
        WebAppInterface(Context c) {
            context = c;
        }

        @JavascriptInterface
        public void triggerToast(String toast) {
            Toast.makeText(context, toast, Toast.LENGTH_SHORT).show();
        }
    }
}
```

Evo HTML koda koji služi za pokretanje koda iz listinga 10.9:

```
<input type="text" name="toastText" id="toastText" />
<button id="btn" onClick="androidToast()">Toast it</button>
```

Sledeća JavaScript funkcija pokreće izvršavanje koda:

```
function androidToast() {
  var input = document.getElementById('toastText');
  Android.triggerToast(input.value);
}
```

Slika 10.5 prikazuje komponentu WebView s tekstom poruke tipa `Toast` čije je prikazivanje pokrenuto iz koda stranice koja je otvorena na uređaju.

Recept: Obrada podataka u formatu JSON

JSON je veoma popularan format za razmenu podataka, naročito u slučaju veb servisa. U paketu `org.json` Android sadrži skup klasa koje možete uvesti u svoj kôd da biste obrađivali JSON podatke.

Da biste započeli obradu, prvo morate napraviti objekat klase JSON, na sledeći način:

```
private JSONObject jsonObject;
```

Slika 10.5 Prikazivanje poruke tipa Toast iz koda stranice koju prikazuje komponenta WebView

Potrebni su vam i podaci u JSON formatu. Sledeća naredba formira znakovni niz s nekoliko podataka u JSON formatu:

```
private String jsonString =
"{\"item\":{\"name\":\"myName\",\"numbers\":[{\"id\":\"1\"},
   {\"id\":\"2\"}]}}";
```

Pošto znakovni niz nije JSON objekat, treba da napravite takav objekat koji sadrži vrednost znakovnog niza. To možete uraditi otprilike ovako:

```
jsonObject = new JSONObject(jsonString);
```

Pošto sada imate upotrebljiv objekat, možete preuzeti podatke iz njega. Ako pozovete metodu `getString()` da biste preuzeli podatke iz „objekta“ sadržanog unutar promenljive `jsonObject`, generisaće se izuzetak tipa `JSONException` jer taj objekat nije znakovni niz. Da biste izdvojili datu vrednost, morate formirati drugi objekat koji sadrži traženu znakovnu vrednost, na sledeći način:

```
JSONObject itemObject = jsonObject.getJSONObject("item");
```

Evo kako ćete učitati vrednost elementa „name“:

```
String jsonName = itemObject.getString("name");
```

Podaci sadržani u elementu „numbers“ promenljive `jsonObject` mogu se iščitati u petlji. To ćete uraditi tako što ćete formirati objekat tipa `JSONArray` i obraditi ga u petlji:

```
JSONArray numbersArray = itemObject.getJSONArray("numbers");

for(int i = 0;i < numbersArray.length();i++){
  numbersArray.getJSONObject(i).getString("id");
}
```

Listing 10.10 prikazuje kako se obrada može ugraditi u aktivnost, a rezultat prikazati pomoću komponente `TextView`. Imajte u vidu sledeće: kada preuzimate JSON podatke sa udaljenog mesta – npr. preko veb servisa, morate koristiti zasebnu klasu ili objekat tipa `AsyncTask` da ne biste blokirali glavnu nit korisničkog interfejsa.

Listing 10.10 src/com/cookbook/parsejson/MainActivity.java

```
package com.cookbook.parsejson;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends Activity {

   TextView tv;
   private JSONObject jsonObject;
   private String jsonString =
"{\"item\":{\"name\":\"myName\",\"numbers\":[{\"id\":\"1\"},{\"id\":\"2\"}]}}";

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      tv = (TextView) findViewById(R.id.tv_main);

      try {
         jsonObject = new JSONObject(jsonString);
         JSONObject itemObject = jsonObject.getJSONObject("item");
         String jsonName = "name: " +itemObject.getString("name");
         JSONArray numbersArray = itemObject.getJSONArray("numbers");
         String jsonIds = "";

         for(int i = 0;i < numbersArray.length();i++){
           jsonIds += "id: " +
              numbersArray.getJSONObject(i).getString("id").toString() + "\n";
         }
```

```
            tv.setText(jsonName+"\n"+jsonIds);

        } catch (JSONException e) {
          e.printStackTrace();
        }
    }
}
```

Recept: Obrada XML podataka

Zvanična dokumentacija Androida preporučuje upotrebu klase `XmlPullParser` za rad sa XML podacima. XML podatke možete pripremiti na bilo koji način koji vam odgovara; u ovom receptu koristićemo jednostavan XML znakovni niz koji sadrži samo jedan čvor. Listing 10.11 prikazuje aktivnost koja učitava ceo XML dokument, uključujući sam čvor i vrednost teksta u njemu, u prikaz TextView.

XML podaci se obrađuju red po red, a metoda `next()` učitava sledeći red. Da biste u XML podacima pronašli traženi čvor, u petlju `while` morate dodati naredbu `if else`.

Listing 10.11 src/com/cookbook/parsexml/MainActivity.java

```
package com.cookbook.parsexml;

import java.io.IOException;
import java.io.StringReader;

import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;
import org.xmlpull.v1.XmlPullParserFactory;

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

public class MainActivity extends Activity {

   TextView tv;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      String xmlOut = "";
      XmlPullParserFactory factory = null;
      try {
        factory = XmlPullParserFactory.newInstance();
      } catch (XmlPullParserException e) {
        e.printStackTrace();
```

```
        }
        factory.setNamespaceAware(true);
        XmlPullParser xpp = null;
        try {
          xpp = factory.newPullParser();
        } catch (XmlPullParserException e) {
          e.printStackTrace();
        }

        try {
          xpp.setInput(new StringReader("<node>This is some text</node>"));
        } catch (XmlPullParserException e) {
          e.printStackTrace();
        }

        int eventType = 0;
        try {
          eventType = xpp.getEventType();
        } catch (XmlPullParserException e) {
          e.printStackTrace();
        }

        while (eventType != XmlPullParser.END_DOCUMENT) {
           if(eventType == XmlPullParser.START_DOCUMENT) {
             xmlOut += "Start of XML Document";
           } else if (eventType == XmlPullParser.START_TAG) {
             xmlOut += "\nStart of tag: "+xpp.getName();
           } else if (eventType == XmlPullParser.END_TAG) {
             xmlOut += "\nEnd of tag: "+xpp.getName();
           } else if (eventType == XmlPullParser.TEXT) {
             xmlOut += "\nText: "+xpp.getText();
           }
           try {
             eventType = xpp.next();
           } catch (XmlPullParserException e) {
             e.printStackTrace();
           } catch (IOException e) {
             e.printStackTrace();
           }
        }
        xmlOut += "\nEnd of XML Document";

        tv.setText(xmlOut);
    }
}
```

Društvene mreže

Twitter je društvena mreža koja omogućava korisnicima da šalju i primaju poruke poznate kao tvitovi (engl. *tweet* – cvrkutanje). Twitter se opisuje kao „SMS na internetu" jer svaki tvit može sadržati najviše 140 znakova (mada se hiperveze konvertuju u kraće oblike i ne računaju se u ograničenje od 140 znakova). Korisnici Twittera mogu da slede tvitove drugih korisnika ili da drugi korisnici slede njihove.

Recept: Učitavanje profila vlasnika uređaja

Počev od API-ja Level 14 (Ice Cream Sandwich), projektanti aplikacija mogu da pristupaju profilu vlasnika uređaja. To je specijalan kontakt koji sadrži podatke u formatu `RawContact`. Da biste učitali profil vlasnika uređaja, u datoteku **AndroidManifest.xml** morate dodati sledeće ovlašćenje:

```
<uses-permission android:name="android.permission.READ_PROFILE" />
```

Naredni blok koda omogućava prisupanje podacima u profilu:

```
// kolone za podatke iz profila vlasnika – objekat RawContact
String[] mProjection = new String[]
    {
          Profile._ID,
          Profile.DISPLAY_NAME_PRIMARY,
          Profile.LOOKUP_KEY,
          Profile.PHOTO_THUMBNAIL_URI
    };

// Učitavanje profila iz izvora Contacts
Cursor mProfileCursor =
    getContentResolver().query(Profile.CONTENT_URI,mProjection,null,null,
      null);
// Postavlja kursor na prvi zapis (umesto na -1)
boolean b = mProfileCursor.moveToFirst();
for(int i = 0, length = mProjection.length;i < length;i++) {
    System.out.println("*** " +
       mProfileCursor.getString(mProfileCursor.getColumnIndex
         (mProjection[i])));
}
```

Imajte u vidu da na mesto na kome se nalazi naredba `System.out.println()` možete ugraditi logiku za obradu podataka iz učitanog profila. Vredi napomenuti i to da se rezultat naredbe prikazuje u prozoru alatke LogCat, uprkos tome što to nije metoda iz `Log.*`.

Recept: Integrisanje Twittera u aplikaciju

Postoje biblioteke iz trećih izvora koje olakšavaju integrisanje Twittera u Android aplikacije (preneto sa lokacije http://dev.twitter.com/pages/libraries#java):

- Twitter4J, čiji je autor Yusuke Yamamoto – Biblioteka Java funkcija za Twitter API, koja je otvorenog koda i podržava upotrebu alatke Maven i platforme Google App Engine; može se koristiti pod licencom BSD.
- Scribe, čiji je autor Pablo Fernandez – OAuth modul za Javu, podržava upotrebu alatke Maven i radi s Facebookom, LinkedInom, Twitterom, Evernoteom, Vimeom i drugim društvenim mrežama.

U ovom receptu koristimo biblioteku Twitter4J čiji je autor Yusuke Yamamoto. Dokumentacija se nalazi na adresi http://twitter4j.org/en/javadoc/overview-summary.html. Recpt omogućava korisniku da se pomoću sistema provere identiteta OAuth prijavi na Twitter i da pošalje poruku (tvit).

Twitter je izmenio svoj sistem utvrđivanja identiteta korisnika tako da sada aplikacije moraju da se registruju kako bi mogle da pristupaju javnom delu. Aplikacija se mora prvo registrovati na https://dev.twitter.com/apps/new. Tokom postupka registrovanja generišu se javni i privatni OAuth ključevi. Pošto će nam trebati u ovom receptu, zabeležite ih negde.

Budući da će ova aplikacija pristupati internetu, potrebno joj je ovlašćenje `INTERNET`. Pošto ćemo takođe proveravati da li je uređaj povezan u mrežu, treba nam i ovlašćenje `ACCESS_NETWORK_STATE`. To ćemo postići tako što ćemo izmeniti datoteku **AndroidManifest.xml** kao što je prikazano u listingu 10.12.

Listing 10.12 AndroidManifest.xml

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

   <uses-sdk
      android:minSdkVersion="9"
      android:targetSdkVersion="17" />

   <uses-permission android:name="android.permission.INTERNET" />
   <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity
         android:name="com.cookbook.tcookbook.MainActivity"
         android:label="@string/app_name" >
         <intent-filter>
```

```
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
         <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <category android:name="android.intent.category.DEFAULT" />
            <category android:name="android.intent.category.BROWSABLE" />
         </intent-filter>
      </activity>
   </application>
</manifest>
```

Ekran aplikacije definisan je u datoteci **activity_main.xml**. Ta datoteka sadrži dugme koje se vidi tokom učitavanja stranice, a zatim se prikazuje i nekoliko drugih dugmadi, natpisa (TextView) i jedno polje tipa `EditText` za unošenje teksta. Obratite pažnju na to da su neke od tih kontrola skrivene jer je za njih zadato `android:visibility="gone"`. Listing 10.13 prikazuje sadržaj datoteke **activity_main.xml**.

Listing 10.13 res/layout/activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >
   tools:context=".MainActivity" >

   <Button android:id="@+id/btnLoginTwitter"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Login with OAuth"
      android:layout_marginLeft="10dip"
      android:layout_marginRight="10dip"
      android:layout_marginTop="30dip"/>

   <TextView android:id="@+id/lblUserName"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:padding="10dip"
      android:layout_marginTop="30dip"/>

   <TextView android:id="@+id/lblUpdate"
      android:text="Enter Your Tweet:"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_marginLeft="10dip"
```

```
        android:layout_marginRight="10dip"
        android:visibility="gone"/>

    <EditText android:id="@+id/txtUpdateStatus"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dip"
        android:visibility="gone"/>

    <Button android:id="@+id/btnUpdateStatus"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Tweet it!"
        android:layout_marginLeft="10dip"
        android:layout_marginRight="10dip"
        android:visibility="gone"/>

    <Button android:id="@+id/btnLogoutTwitter"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Logout/invalidate OAuth"
        android:layout_marginLeft="10dip"
        android:layout_marginRight="10dip"
        android:layout_marginTop="50dip"
        android:visibility="gone"/>
</LinearLayout>
```

U aplikaciji koristimo jednu aktivnost i dve klase: jednu za praćenje stanja veze s mrežom i drugu za upozoravajuću poruku ako korisnik zada pogrešne OAuth ključeve za aplikaciju.

U glavnoj aktivnosti definisano je više konstanti. Među njima su i vrednosti za OAuth komponente „Consumer key“ i „Consumer secret“. Aplikacija proverava da li je uređaj povezan u mrežu i da li korisnik može da dopre do Twittera. Osim toga, aplikacija registruje više objekata klase `OnClickListener` koji pokreću logiku za obradu događaja kao što su prijavljivanje na Twitter, odjavljivanje i ažuriranje kad korisnik pritisne odgovarajuće dugme.

Pošto Twitter proverava identitet korisnika, informacije koje on vraća čuvaju se u obliku preferenci aplikacije i ponovo koriste kada sledeći put korisnik pokuša da se prijavi u aplikaciju. Poruke (tvitovi) koje korisnik šalje prosleđuju se pomoću objekta tipa AsyncTask u pozadinsku nit na obradu.

Listing 10.14 prikazuje ceo kôd aktivnosti.

Listing 10.14 src/com/cookbook/tcookbook/MainActivity.java

```
package com.cookbook.tcookbook;

import twitter4j.Twitter;
import twitter4j.TwitterException;
import twitter4j.TwitterFactory;
import twitter4j.User;
import twitter4j.auth.AccessToken;
import twitter4j.auth.RequestToken;
import twitter4j.conf.Configuration;
import twitter4j.conf.ConfigurationBuilder;
import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.content.pm.ActivityInfo;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Build;
import android.os.Bundle;
import android.os.StrictMode;
import android.text.Html;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {

   // Ovu vrednost treba da zameni vaš "Consumer key"
   static String TWITTER_CONSUMER_KEY = "01189998819991197253";
   // Ovu vrednost treba da zameni vaš "Consumer secret"
   static String TWITTER_CONSUMER_SECRET =
      "616C6C20796F7572206261736520617265206562656C6F6E6720746F207573";

   static String PREFERENCE_NAME = "twitter _ oauth";
   static final String PREF_KEY_OAUTH_TOKEN = "oauth_token";
   static final String PREF_KEY_OAUTH_SECRET = "oauth_token_secret";
   static final String PREF_KEY_TWITTER_LOGIN = "isTwitterLoggedIn";

   static final String TWITTER_CALLBACK_URL = "oauth://tcookbook";

   static final String URL_TWITTER_AUTH = "auth_url";
   static final String URL_TWITTER_OAUTH_VERIFIER = "oauth_verifier";
   static final String URL_TWITTER_OAUTH_TOKEN = "oauth_token";
```

```
Button btnLoginTwitter;
Button btnUpdateStatus;
Button btnLogoutTwitter;
EditText txtUpdate;
TextView lblUpdate;
TextView lblUserName;

ProgressDialog pDialog;

private static Twitter twitter;
private static RequestToken requestToken;

private static SharedPreferences mSharedPreferences;

private ConnectionDetector cd;

AlertDialogManager adm = new AlertDialogManager();

@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   // Koristi se na Androidu 2.3+
   if (Build.VERSION.SDK_INT > Build.VERSION_CODES_GINGERBREAD) {
      StrictMode.ThreadPolicy policy =
         new StrictMode.ThreadPolicy.Builder().permitAll().build();
      StrictMode.setThreadPolicy(policy);
   }

   setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);

   cd= new ConnectionDetector(getApplicationContext());
   if (!cd.isConnectingToInternet()) {
      adm.showAlertDialog(MainActivity.this, "Greška, veza sa internetom ",
         "Nije uspostavljena veza sa internetom", false);
      return;
   }

   if(TWITTER_CONSUMER_KEY.trim().length() == 0 ||
      TWITTER_CONSUMER_SECRET.trim().length() == 0){
     adm.showAlertDialog(MainActivity.this,
         "OAuth žetoni za Twitter",
         "Pribavite prvo OAuth žetone za Twitter!", false);
     return;
   }

   btnLoginTwitter = (Button) findViewById(R.id.btnLoginTwitter);
   btnUpdateStatus = (Button) findViewById(R.id.btnUpdateStatus);
```

```
        btnLogoutTwitter = (Button) findViewById(R.id.btnLogoutTwitter);
        txtUpdate = (EditText) findViewById(R.id.txtUpdateStatus);
        lblUpdate = (TextView) findViewById(R.id.lblUpdate);
        lblUserName = (TextView) findViewById(R.id.lblUserName);

        mSharedPreferences = getApplicationContext().
             getSharedPreferences("MyPref", 0);

        btnLoginTwitter.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View arg0) {
             // Poziva Twitterovu funkciju za prijavljivanje
             loginToTwitter();
           }
        });

        btnUpdateStatus.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View v) {
             String status = txtUpdate.getText().toString();

             if (status.trim().length() > 0) {
               new updateTwitterStatus().execute(status);
             } else {
               Toast.makeText(getApplicationContext(),
                    "Upišite statusnu poruku ", Toast.LENGTH_SHORT).show();
             }
           }
        });

        btnLogoutTwitter.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View arg0) {
             // Poziva Twitterovu funkciju za odjavljivanje
             logoutFromTwitter();
           }
        });

        if (!isTwitterLoggedInAlready()) {
           Uri uri = getIntent().getData();

           if (uri != null && uri.toString().startsWith(TWITTER_CALLBACK_URL)) {
             String verifier = uri.getQueryParameter(URL_TWITTER_OAUTH_VERIFIER);

             try {
             AccessToken accessToken = twitter.getOAuthAccessToken(requestToken,
➥verifier);
```

```
            Editor e = mSharedPreferences.edit();

            e.putString(PREF_KEY_OAUTH_TOKEN, accessToken.getToken());
            e.putString(PREF_KEY_OAUTH_SECRET,accessToken.getTokenSecret());
            e.putBoolean(PREF_KEY_TWITTER_LOGIN, true);
            e.commit();

//          Log.e("Twitter OAuth Token", "> " + accessToken.getToken());

            btnLoginTwitter.setVisibility(View.GONE);
            lblUpdate.setVisibility(View.VISIBLE);
            txtUpdate.setVisibility(View.VISIBLE);
            btnUpdateStatus.setVisibility(View.VISIBLE);
            btnLogoutTwitter.setVisibility(View.VISIBLE);

            long userID = accessToken.getUserId();
            User user = twitter.showUser(userID);
            String username = user.getName();

            lblUserName.setText(Html.fromHtml("<b>Dobrodošli " + username +
               "</b>"));
          } catch (Exception e) {
            Log.e("***Greška pri prijavljivanju na Twitter: ",
               e.getMessage());
          }
       }
    }
}

private void loginToTwitter() {
   if (!isTwitterLoggedInAlready()) {
      ConfigurationBuilder builder = new ConfigurationBuilder();
      builder.setOAuthConsumerKey(TWITTER_CONSUMER_KEY);
      builder.setOAuthConsumerSecret(TWITTER_CONSUMER_SECRET);
      Configuration configuration = builder.build();

      TwitterFactory factory = new TwitterFactory(configuration);
      twitter = factory.getInstance();

      if(!(Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB)) {
         try {
            requestToken = twitter.getOAuthRequestToken(TWITTER_CALLBACK_URL);
            this.startActivity(new Intent(Intent.ACTION_VIEW,
               Uri.parse(requestToken.getAuthenticationURL())));
         } catch (TwitterException e) {
           e.printStackTrace();
         }
         } else {
         new Thread(new Runnable() {
```

```
                public void run() {
                   try {
                     requestToken = twitter.getOAuthRequestToken
                        (TWITTER_CALLBACK_ URL);
                     MainActivity.this.startActivity(new Intent(Intent.ACTION_
                        VIEW,
                      Uri.parse(requestToken.getAuthenticationURL())));
                   } catch (TwitterException e) {
                     e.printStackTrace();
                   }
                 }
              }).start();
          }
       } else {
         Toast.makeText(getApplicationContext(),"Already logged into Twitter",
             Toast.LENGTH_LONG).show();
       }
    }

    class updateTwitterStatus extends AsyncTask<String, String, String> {
       @Override
       protected void onPreExecute() {
         super.onPreExecute();
         pDialog = new ProgressDialog(MainActivity.this);
         pDialog.setMessage("Updating to Twitter...");
         pDialog.setIndeterminate(false);
         pDialog.setCancelable(false);
         pDialog.show();
       }

       protected String doInBackground(String... args) {
//           Log.d("*** Tekst tvita: ",args[0]);
          String status = args[0];
          try {
            ConfigurationBuilder builder = new ConfigurationBuilder();
            builder.setOAuthConsumerKey(TWITTER_CONSUMER_KEY);
            builder.setOAuthConsumerSecret(TWITTER_CONSUMER_SECRET);

            String access_token =
                mSharedPreferences.getString(PREF_KEY_OAUTH_TOKEN, "");
            String access_token_secret =
                mSharedPreferences.getString(PREF_KEY_OAUTH_SECRET, "");

            AccessToken accessToken =
                new AccessToken(access_token, access_token_secret);
            Twitter twitter =
                new TwitterFactory(builder.build()).getInstance(accessToken);

            twitter4j.Status response = twitter.updateStatus(status);
```

```
//          Log.d("***Status ažuriranja: ",response.getText());
         } catch (TwitterException e) {
           Log.d("*** Twitter, greška pri ažuriranju: ", e.getMessage());
         }
         return null;
      }

      protected void onPostExecute(String file_url) {
          pDialog.dismiss();
          runOnUiThread(new Runnable() {
            @Override
            public void run() {
               Toast.makeText(getApplicationContext(),
                  "Status uspešno tvitnut", Toast.LENGTH_SHORT).show();
               txtUpdate.setText("");
              }
            });
         }
      }

      private void logoutFromTwitter() {
         Editor e = mSharedPreferences.edit();
         e.remove(PREF_KEY_OAUTH_TOKEN);
         e.remove(PREF_KEY_OAUTH_SECRET);
         e.remove(PREF_KEY_TWITTER_LOGIN);
         e.commit();

         btnLogoutTwitter.setVisibility(View.GONE);
         btnUpdateStatus.setVisibility(View.GONE);
         txtUpdate.setVisibility(View.GONE);
         lblUpdate.setVisibility(View.GONE);
         lblUserName.setText("");
         lblUserName.setVisibility(View.GONE);

         btnLoginTwitter.setVisibility(View.VISIBLE);
      }

      private boolean isTwitterLoggedInAlready() {
         return mSharedPreferences.getBoolean(PREF_KEY_TWITTER_LOGIN, false);
      }

      protected void onResume() {
         super.onResume();
      }
}
```

Više informacija o korišćenju biblioteke Twitter4j naći ćete na sledećim adresama:

- www.androidhive.info/2012/09/android-twitter-oauth-connect-tutorial/; autor je Ravi Tamada
- http://blog.doityourselfandroid.com/2011/08/08/improved-twitter-oauth-android/; lokacija Do-it-yourself Android
- http://davidcrowley.me/?p=410; autor je David Crowley
- https://tutsplus.com/tutorials/?q=true&filter_topic=90; autor je Sue Smith
- http://blog.blundell-apps.com/sending-a-tweet/; autor je Blundell

Recept: Integrisanje Facebooka u aplikaciju

Facebook se brzo menjao poslednjih nekoliko godina i ostaje jedna od najpopularnijih društvenih mreža. Facebookov tim je nedavno pročistio svoju dokumentaciju da bi pomogao projektantima aplikacija. Zvaničnu dokumentaciju možete naći na https://developers.facebook.com/docs/getting-started/facebook-sdk-for-android/3.0/.

Da biste započeli razvoj aplikacija za Facebook, prvo preuzmite paket Facebook SDK i Facebook Android (APK) s lokacije https://developers.facebook.com/resources/facebook-android-sdk-3.0.zip. Svrha APK-a je da omogući utvrđivanje identiteta korisnika bez upotrebe prikaza WebView. Ako je Facebook aplikacija već instalirana na telefon, nije potrebno instaliranje APK datoteke.

Dalje, dodajte Facebook SDK u okruženje Eclipse kao bibliotečki projekat. To ćete uraditi ako izaberete **File → Import**, a zatim **General → Existing Projects into Workspace**. Imajte u vidu da Facebook upozorava da se ne koriste opcije „Copy projects into workspace“ zato što se tako mogu formirati pogrešne putanje u sistemu datoteka, što će sprečiti SDK da radi kako treba.

Pošto uvezete Facebook SDK, imaćete na raspolaganju uzorke projekata za eksperimentisanje. Imajte u vidu da je za većinu projekata potrebno generisati heš-ključ koji će služiti za potpisivanje aplikacija koje autori mogu zatim da dodaju svom profilu Facebook projektanta radi brzog pristupanja projektima u SDK-u.

Ključ se generiše pomoću alatke `keytool` koja je sastavni deo Jave. Otvorite prozor terminala ili komandni prozor i otkucajte sledeće da biste generisali ključ:

OS X:

```
keytool -exportcert -alias androiddebugkey -keystore ~/.android/debug.
keystore | [ccc]openssl sha1 -binary | openssl base64
```

Windows:

```
keytool -exportcert -alias androiddebugkey -keystore %HOMEPATH%\.android\
debug.keystore [ccc]| openssl sha1 -binary | openssl base64
```

Celu komandu treba otkucati u jednom redu, mada se ona u prozoru terminala ili komandnom prozoru može prikazati prelomljena u više redova. Pošto izvršite komandu, trebalo bi da se pojavi zahtev da unesete lozinku. Upišite **android**. Pošto se ključ uspešno generiše, prikazuje se njegova vrednost. Ako se pojavi greška „`'keytool' is not recognized as an internal or external command . . .`“, pređite u direktorijum **bin** JRE instalacije i pokušajte ponovo.

Ukoliko se pojavi ista greška za „openssl“, preuzmite OpenSSL sa adrese http://code.google.com/p/openssl-for-windows/. Ako se i nakon toga pojavljuju greške, proverite da li su direktorijumi **bin** dodati u sistemsku putanju, ili da li su ti direktorijumi zadati eksplicitno, a ne sa **%HOMEPATH%**.

Ako se za razvijanje aplikacija koristi više računara, za svaki od njih mora se generisati zaseban heš i dodati u projektantov profil na adresi https://developers.facebook.com/.

Pošto to obavite, pokrenite primere aplikacije i prijavite se na Facebook iz njih. Primer projekta, nazvan **HelloFacebookSample**, ilustruje kako se pristupa datom profilu, ažurira status, pa čak u kako se šalju fotografije na Facebook.

Poslednji korak postupka integrisanja aplikacije s Facebookom jeste izrada Facebook aplikacije koja će zatim biti povezana s Android aplikacijom preko generisnog heša ključa. Time se ostvaruje integracija i omogućava korisnicima da potvrde svoj identitet kada koriste aplikaciju.

Veb lokacija namenjena projektantima aplikacija sadrži odličan opis svih delova postupka koji je potreban da biste krenuli. Obavezno pročitajte zvanično uputstvo Scrumptious, koje ćete naći na adresi http://developers.facebook.com/docs/tutorials/androidsdk/3.0/scrumptious/.