# ANDROID platforma
## Messaging & Content Providers

Mobilni i distribuirani informacioni sistemi

*Mr Bratislav Predić*

2012. godina

# Android i razmena poruka

- Razmena poruka (SMS i MMS) je najstarija dodatna usluga mobilne telefonije
- SMS omogućava razmenu kratkih poruka direktno između telefona, bez treće strane (servera)
  - **Tekstualne poruke** – najčešće namenjene korisnicima
  - **Binarne poruke** – za direktnu komunikaciju između aplikacija
- MMS poruke omogućavaju razmenu poruka koje mnogo više liče na Email poruke
  - U attachment-u: slike, zvuk, video...

# Android i razmena poruka

- Za rad sa SMS porukama na Android platformi se koristi *SMSManager*
- *SMSManager* omogućava
  - Slanje tekstualnih poruka
  - Prijem poruka (reakciju aplikacije na dolaznu poruku)
  - Korišćenje poruka kao sloja za transport podataka
- SMS poruke možemo slati korišćenjem:
  - *SMSManager*-a
  - Slanjem odgovarajućeg *Intent*-a
- Komunikacija SMS porukama je asinhrona, nepouzdana i tipično traje dugo

# SMS - Intent i native klijent

- Kada je potrebno poslati SMS poruku sa predefinisanim sadržajem ili na predefinisano odredište (error report i sl.) dobro je koristiti sistemski SMS klijent
- U tu svrhu se koristi *Intent* sa akcijom *Intent.ACTION_SENDTO* i metoda *startActivity*
- *Intent* data treba da sadrži odredište sa URI prefiksom *sms:*
- Sadržaj poruke se specificira kao *Extra* sa nazicom *sms_body*

# Native messaging client primeri

- Slanje SMS poruke korišćenjem native klijent aplikacije

```
Intent smsIntent = new Intent(Intent.ACTION_SENDTO,
                 Uri.parse("sms:55512345"));
smsIntent.putExtra("sms_body", "Press send to send me");
startActivity(smsIntent);
```

- Slanje MMS poruke native klijentom

```
// Get the URI of a piece of media to attach.
Uri attached_Uri = Uri.parse("content://media/external/images/media/1");
Intent mmsIntent = new Intent(Intent.ACTION_SEND, attached_Uri);
mmsIntent.putExtra("sms_body", "Please see the attached image");
mmsIntent.putExtra("address", "07912355432");
mmsIntent.putExtra(Intent.EXTRA_STREAM, attached_Uri);
mmsIntent.setType("image/png");
startActivity(mmsIntent);
```

- Ovaj *Intent* mogu da obrade različite aplikacije: Email, Gmail, SMS, korisnik ima izbor

# Manuelni rad sa porukama

- Kompletna funkcionalnost rada sa porukama je omogućena klasom *SMSManager* iz paketa *android.telephony*
- Instanciranje *SMSManager*-a

```
SmsManager smsManager = SmsManager.getDefault();
```

- Obzirom da je u pitanju bezbednosno kritična operacija mora biti prijavljena u manifest-u aplikacije

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

- Slanje poruke *SMSManager*-om

```
String sendTo = "5551234";
String myMessage = "Android supports programmatic SMS messaging!";
smsManager.sendTextMessage(sendTo, null, myMessage, null, null);
```

# Manuelni rad sa porukama

```
smsManager.sendTextMessage(sendTo, null, myMessage, null, null);
```

- Dodatni argumenti
  - Drugi – SMS service center – null koristi default
  - Četvrti i peti – Intent-i koji se emituju po uspešnom slanju i prijemu poruke (receipt)
- Za obradu ovih Intent-a se registruju BroadcastReceiver-i
- Moguće je smati SMS poruke između više instanci Android emulatora – kao broj telefona se koristi broj port-a

# Praćenje procesa slanja poruke

- Intent kao četvrti parametar se emituje kada je poruka uspešno poslata ili slanje nije uspelo
- Stanje je u result code-u BroadcastReceiver-a
  - *Activity.RESULT_OK* – uspešno poslat SMS
  - *SmsManager.RESULT_ERROR_GENERIC_FAILURE* – došlo je do nepoznate greške prilikom slanja
  - *SmsManager.RESULT_ERROR_RADIO_OFF* – nemoguće slanje zbog isključenog GSM predajnika
  - *SmsManager.RESULT_ERROR_NULL_PDU* – Protocol description unit failure

# Praćenje procesa slanja poruke

```
String SENT_SMS_ACTION = "SENT_SMS_ACTION";
String DELIVERED_SMS_ACTION = "DELIVERED_SMS_ACTION";
Intent sentIntent = new Intent(SENT_SMS_ACTION);
PendingIntent sentPI = PendingIntent.getBroadcast(getApplicationContext(),
                                    0,sentIntent,0);
// Create the deliveryIntent parameter
Intent deliveryIntent = new Intent(DELIVERED_SMS_ACTION);
PendingIntent deliverPI = PendingIntent.getBroadcast(getApplicationContext(),
                                                    0,deliveryIntent,0);
// Register the Broadcast Receivers
registerReceiver(new BroadcastReceiver() {
  @Override
  public void onReceive(Context _context, Intent _intent)
  {
    switch (getResultCode()) {
      case Activity.RESULT_OK:
        [ . . . send success actions . . . ]; break;
      case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
        [ . . . generic failure actions . . . ]; break;
      case SmsManager.RESULT_ERROR_RADIO_OFF:
        [ . . . radio off failure actions . . . ]; break;
      case SmsManager.RESULT_ERROR_NULL_PDU:
        [ . . . null PDU failure actions . . . ]; break;
    }
  }
}, new IntentFilter(SENT_SMS_ACTION));
...
```

# Praćenje procesa slanja poruke

```
registerReceiver(new BroadcastReceiver() {
  @Override
  public void onReceive(Context _context, Intent _intent)    {
    [ . . . SMS delivered actions . . . ]
  }
}, new IntentFilter(DELIVERED_SMS_ACTION));
// Send the message
smsManager.sendTextMessage(sendTo, null, myMessage, sentPI, deliverPI);
```

- *PendingIntent* – validan čak i kad aplikacija koja ga je kreirala više nije aktivna
- *PendingIntent* se izvršava u kontekstu (sa pravima pristupa) aplikacije sa kojim je kreiran

# Ograničenje veličine poruke

- SMS tekst poruke su ograničene na 160 karaktera (bajtova)
- Duže poruke moraju da se razbiju na više manjih
- *SMSManager* sadrži metodu *divideMessage()* koja za ulazni string vraća niz poruka
- Za slanje niza poruka se koristi metoda *sendMultipartTextMessage()* klase *SMSManager*
- Umesto *Intent*-a se koriste *ArrayList* intenta, za svaku poruku posebno

# Primer slanja multipart poruke

```
ArrayList<String> messageArray = smsManager.divideMessage(myMessage);
ArrayList<PendingIntent> sentIntents = new ArrayList<PendingIntent>();
for (int i = 0; i < messageArray.size(); i++)
  sentIntents.add(sentPI);
smsManager.sendMultipartTextMessage(sendTo,
                                    null,
                                    messageArray,
                                    sentIntents, null);
```

- Za aplikacije je često korisnije da razmenjuju podatke korišćenjem binarnih data SMS poruka
- Klasa *SMSManager* ima metodu *sendDataMessage()*
- Binarne SMS poruke imaju i broj porta na koji se šalju (aplikacija na prijemnoj strani se vezuje za port)

# Binarne SMS poruke

- Primer slanja binarne SMS poruke

```
Intent sentIntent = new Intent(SENT_SMS_ACTION);
PendingIntent sentPI = PendingIntent.getBroadcast(getApplicationContext(),
                                    0, sentIntent, 0);
short destinationPort = 80;
byte[] data = [ . . . your data . . . ];
smsManager.sendDataMessage(sendTo, null, destinationPort,
                                    data, sentPI, null);
```

- Broj porta se kodira na početku poruke pa oduzima par bajtova od sadržaja

# Dolazne SMS poruke

- Kada Android uređaj primi SMS poruku emituje se broadcast *Intent* sa akcijom *android.provider.Telephony.SMS_RECEIVED*
- Kao i za slanje, aplikacija u manifestu mora da prijavi prijem SMS poruke

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

- SMS broadcast Intent u svom Extras delu pod imenom *pdu* sadrži niz *SmsMessage* objekata (u PDU obliku – niz bajtova)
- Za konverziju PDU u *SmsMessage* objekte koristimo *SmsMessage.createFromPdu()*

# Dolazne SMS poruke

- Primer preuzimanja *SmsMessage* iz Intent-a

```
Bundle bundle = intent.getExtras();
if (bundle != null) {
  Object[] pdus = (Object[]) bundle.get("pdus");
  SmsMessage[] messages = new SmsMessage[pdus.length];
  for (int i = 0; i < pdus.length; i++)
    messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
}
```

- Svaki objekat *SmsMessage* sadrži broj telefona sa kog je poslat, vreme i sadržaj poruke
- Primer upotrebe: broadcast receiver koji će svaku poruku koja sadrži ključnu reč @echo istu poruku vratiti na broj sa kog je poslata

# SMS BroadcastReceiver

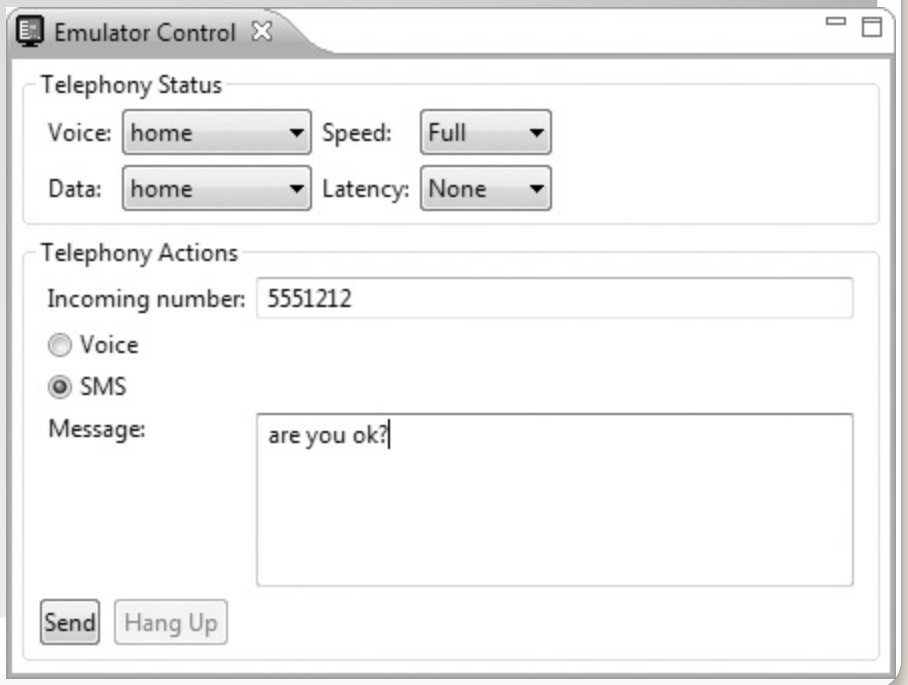
```
public class IncomingSMSReceiver extends BroadcastReceiver {
  private static final String queryString = "@echo";
  private static final String SMS_RECEIVED =
                          "android.provider.Telephony.SMS_RECEIVED";

  public void onReceive(Context _context, Intent _intent) {
    if (_intent.getAction().equals(SMS_RECEIVED)) {
      SmsManager sms = SmsManager.getDefault();
      Bundle bundle = _intent.getExtras();
      if (bundle != null) {
        Object[] pdus = (Object[]) bundle.get("pdus");
        SmsMessage[] messages = new SmsMessage[pdus.length];
        for (int i = 0; i < pdus.length; i++)
          messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
          for (SmsMessage message : messages) {
            String msg = message.getMessageBody();
            String to = message.getOriginatingAddress();
            if (msg.toLowerCase().startsWith(queryString)) {
              String out = msg.substring(queryString.length());
              sms.sendTextMessage(to, null, out, null, null);
            }
          }
        }
      }
    }
  }
```

# SMS BroadcastReceiver

- Registrujemo kreirani BroadcastReceiver

```
final String SMS_RECEIVED = "android.provider.Telephony.SMS_RECEIVED";
IntentFilter filter = new IntentFilter(SMS_RECEIVED);
BroadcastReceiver receiver = new IncomingSMSReceiver();
registerReceiver(receiver, filter);
```

- SMS poruke mogu da se šalju iz jednog emulatora u drugi korišćenjem aplikacije kao na realnom telefonu ili
- Može da se koristi DDMS alat za poruke



# SMS BroadcastReceiver

- Podaci iz binarne SMS poruke se preuzimaju na sličan način
- Koristimo *getUserData()* metod

```
byte[] data = msg.getUserData();
```

# Android ContentProvider

- ContentProvider je mehanizam razmene podataka između različitih aplikacija
- Za identifikaciju se koristi URI mehanizam, šema *content://*
- Na ovaj način je obezbeđen univerzalan interfejs za pristup podacima bez obzira na implementaciju perzistencije tih podataka
- Kroz ContentProvider mehanizam je moguće gledati, menjati, brisati i dodavati slogove
- Za ContentProvider mehanizam važi sistem dozvola

# Kreiranje ContentProvider-a

- Kako bi kreirali novi ContentProvider nasleđujemo apstraktnu klasu *ContentProvider*
- Kreiranje i inicijalizacija izvora podataka (perzistentnog storage-a) treba uraditi i override-ovanoj metodi *onCreate()*

```
import android.content.*;
import android.database.Cursor;
import android.net.Uri;
import android.database.SQLException;

public class MyProvider extends ContentProvider {
  @Override
  public boolean onCreate() {
    // TODO Construct the underlying database.
    return true;
  }
}
```

# Kreiranje ContentProvider-a

- ContentProvider treba da ima javni statički property *CONTENT_URI* po kome se taj provider identifikuje
- Ovaj identifikator mora biti jedinstven pa je dobra praksa da se koristi naziv paketa

```
content://com.paad.provider.myapp/elements
content://com.paad.provider.myapp/elements/5
```

- Dve forme URI-a
  - Sve stavke tog tipa (prvi primer)
  - Stavka tog tipa na određenom indeksu (drugi primer)

# Kreiranje ContentProvider-a

- ContentProvider treba da ima javni statički property *CONTENT_URI* po kome se taj provider identifikuje
- Ovaj identifikator mora biti jedinstven pa je dobra praksa da se koristi naziv paketa

```
content://com.paad.provider.myapp/elements
content://com.paad.provider.myapp/elements/5
```

- Dve forme URI-a
  - Sve stavke tog tipa (prvi primer)
  - Stavka tog tipa na određenom indeksu (drugi primer)

# Kreiranje ContentProvider-a

- Za prepoznavanje URI zahteva postoji helper klasa *UriMatcher*

```
public class MyProvider extends ContentProvider {
  private static final String myURI =
                          "content://com.paad.provider.myapp/items";
  public static final Uri CONTENT_URI = Uri.parse(myURI);
  @Override
  public boolean onCreate() {
    // TODO: Construct the underlying database.
    return true;
  }
  private static final int ALLROWS = 1;
  private static final int SINGLE_ROW = 2;
  private static final UriMatcher uriMatcher;

  static {
    uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    uriMatcher.addURI("com.paad.provider.myApp", "items", ALLROWS);
    uriMatcher.addURI("com.paad.provider.myApp", "items/#", SINGLE_ROW);
  }
}
```

# Kreiranje ContentProvider-a

- Upiti i transakcije se omogućavaju implementacijom metoda *delete, insert, update* i *query*
- Najčešći scenario je da se kao mehanizam perzistencije koristi SQLite baza, ali može biti bilo kakav mehanizam koji i ne mora biti perzistentan
- Treba definisati i MIME tip za podatke koje provider vraća

```
@Override
public String getType(Uri _uri) {
  switch (uriMatcher.match(_uri)) {
    case ALLROWS: return "vnd.paad.cursor.dir/myprovidercontent";
    case SINGLE_ROW: return "vnd.paad.cursor.item/myprovidercontent";
    default: throw new IllegalArgumentException("Unsupported URI: " + _uri);
  }
}
```

# Kreiranje ContentProvider-a

- Interfejsne metode

```
@Override
public Cursor query(Uri uri, String[] projection, String selection,
                    String[] selectionArgs, String sort) {
  switch (uriMatcher.match(uri)) {
    case SINGLE_ROW :
    // rowNumber = uri.getPathSegments().get(1));
  }
  return null;
}
@Override
public Uri insert(Uri _uri, ContentValues _initialValues) {
  long rowID = [ ... Add a new item ... ]
  if (rowID > 0) {
    return ContentUris.withAppendedId(CONTENT_URI, rowID);
  }
  throw new SQLException("Failed to add new item into " + _uri);
}
@Override
public int delete(Uri uri, String where, String[] whereArgs) {
  switch (uriMatcher.match(uri)) {
    case ALLROWS:
    case SINGLE_ROW:
    default: throw new IllegalArgumentException("Unsupported URI:" + uri);
  }
}
```

# Kreiranje ContentProvider-a

- Interfejsne metode (cont.)

```
@Override
public int update(Uri uri, ContentValues values, String where, String[]
                                                              whereArgs) {
  switch (uriMatcher.match(uri)) {
    case ALLROWS:
    case SINGLE_ROW:
    default: throw new IllegalArgumentException("Unsupported URI:" + uri);
  }
}
```

- Registracija kreiranog provider-a u manifestu aplikacije

```
<provider android:name="MyProvider"
        android:authorities="com.paad.provider.myapp"/>
```

# Korišćenje ContentProvider-a

- Koristi se klasa *ContentResolver*
- Ova klasa postoji u aplikacionom kontekstu

```
ContentResolver cr = getContentResolver();
```

- Rad sa *ContentProvider*-ima je sličan kao sa bazom
- Query vraća kursor
- Query metoda *ContentResolver*-a ima argumente slične kao upit nad bazom
  - URI ContentProvider-a koji koristimo
  - Lista kolona koje želimo u rezultatu
  - Where klauzula za filtriranje sa ? Parametrima
  - Lista argumenata koji menjaju ?
  - String koji definiše uređenje rezultata

# Korišćenje ContentProvider-a

- Primer query-a

```
ContentResolver cr = getContentResolver();
// Return all rows
Cursor allRows = cr.query(MyProvider.CONTENT_URI, null, null, null, null);
// Return all columns for rows where column 3 equals a set value
// and the rows are ordered by column 5.
String where = KEY_COL3 + "=" + requiredValue;
String order = KEY_COL5;
Cursor someRows = cr.query(MyProvider.CONTENT_URI, null, where, null, order);
```

- Dodavanje novih stavki
  - *insert* – dodaje jedan novi red
  - *bulkInsert* – dodaje niz novih redova
- ds

# Korišćenje ContentProvider-a

- Primer dodavanja redova

```
// Get the Content Resolver
ContentResolver cr = getContentResolver();

// Create a new row of values to insert.
ContentValues newValues = new ContentValues();

// Assign values for each row.
newValues.put(COLUMN_NAME, newValue);
[ ... Repeat for each column ... ]
Uri myRowUri = cr.insert(MyProvider.CONTENT_URI, newValues);

// Create a new row of values to insert.
ContentValues[] valueArray = new ContentValues[5];

// TODO: Create an array of new rows
int count = cr.bulkInsert(MyProvider.CONTENT_URI, valueArray);
```

# Korišćenje ContentProvider-a

- Brisanje redova u ContentPRovider-u

```
ContentResolver cr = getContentResolver();
// Remove a specific row.
cr.delete(myRowUri, null, null);
// Remove the first five rows.
String where = "_id < 5";
cr.delete(MyProvider.CONTENT_URI, where, null);
```

- Možemo brisati konkretan red na osnovu njegovog ID-ja
- Možemo dodati where klauzulu kao uslov brisanja

# Korišćenje ContentProvider-a

- Izmene redova u ContentProvider-u
- Update metoda kao parametra uzima
  - URI
  - *ContentValues* objekat koji mapira imena kolona sa podacima koje unosimo
  - *Where* klauzulu
- Metoda vraća broj izmenjenih redova

```
// Create a new row of values to insert.
ContentValues newValues = new ContentValues();
// Create a replacement map, specifying which columns you want to
// update, and what values to assign to each of them.
newValues.put(COLUMN_NAME, newValue);
// Apply to the first 5 rows.
String where = "_id < 5";
getContentResolver().update(MyProvider.CONTENT_URI, newValues, where, null);
```

# Korišćenje ContentProvider-a

- Datoteke i ContentProvider-i
- Sa datotekama se ne radi kao sa raw blob-ovima već se koristi URI datoteke
- Za pristup samom sadržaju datoteke se koriste metode
  - *openOutputStream()*
  - *openInputStream()*

```
// Insert a new row into your provider, returning its unique URI.
Uri uri = getContentResolver().insert(MyProvider.CONTENT_URI, newValues);
try {
  // Open an output stream using the new row's URI.
  OutputStream outStream = getContentResolver().openOutputStream(uri);
  // Compress your bitmap and save it into your provider.
  sourceBitmap.compress(Bitmap.CompressFormat.JPEG, 50, outStream);
} catch (FileNotFoundException e) { }
```

# Primer sistemskog ContentProvider-a

- Želimo pregled svih postojećih kontakata na sistemu

```
//Here is the button to click for viewing the contacts
Button view = (Button)findViewById(R.id.viewButton);
…
//The method / class that gets invoked when the View button is clicked
view.setOnClickListener(new OnClickListener() {
    public void onClick(View v){
       displayContacts();
       Log.i("NativeContentProvider", "Completed Displaying Contact list");
    }
});
```

- *Activity* ima helper metodu za rad sa kursorima *(managedQuery)*
  - Aktivnost automatski briše kursor kada više nije potreban
  - Eliminiše potencijalno curenje resursa

# Primer sistemskog ContentProvider-a

```
//Here is the displayContacts() method
private void displayContacts() {
  String[] columns = new String[] {People.NAME,People.NUMBER};
  Uri mContacts = People.CONTENT_URI;
  Cursor mCur = managedQuery(mContacts, // Contact URI
                 columns,    // Which columns to return
                 null,       // Which rows to return
                 null,       // Where clause parameters
                 null        // Order by clause
                 );
  if (mCur.moveToFirst()) {
    String name = null;
    String phoneNo = null;
    do {
      name = mCur.getString(mCur.getColumnIndex(People.NAME));
      phoneNo = mCur.getString(mCur.getColumnIndex(People.NUMBER));
      Toast.makeText(NativeContentProvider.this, name + " " + phoneNo,
                          Toast.LENGTH_SHORT).show();
    } while (mCur.moveToNext());
  }
}
```

# Primer sistemskog ContentProvider-a

- Dodajemo novi kontakt u sistem

```
private void createContact(String name, String phone) {
  ContentValues contact = new ContentValues();
  contact.put(People.NAME, name);
  insertUri = getContentResolver().insert(People.CONTENT_URI, contact);
  Log.d(getClass().getSimpleName(),insertUri.toString());
  Uri phoneUri = Uri.withAppendedPath(insertUri,
                                      People.Phones.CONTENT_DIRECTORY);
  contact.clear();
  contact.put(People.Phones.TYPE, People.TYPE_MOBILE);
  contact.put(People.NUMBER, phone);
  updateUri = getContentResolver().insert(phoneUri, contact);
  Toast.makeText(NativeContentProvider.this, "Created a new contact: " + name
                          + " " + phone, Toast.LENGTH_SHORT).show();
  Log.d(getClass().getSimpleName(),updateUri.toString());
}
```

- Koristimo *ContentResolver* i *ContentValues*

# Primer sistemskog ContentProvider-a

```
private void updateContact(String phone) {
  if (updateUri != null) {
    ContentValues newPhone = new ContentValues();
    newPhone.put(People.Phones.TYPE, People.TYPE_MOBILE);
    newPhone.put(People.NUMBER, phone);
    getContentResolver().update(updateUri, newPhone, null,null);
    Log.i(getClass().getSimpleName(), "Updated the phone number");
}

private void deleteContact() {
  if (updateUri != null) {
    getContentResolver().delete(insertUri, null, null);
    Toast.makeText(NativeContentProvider.this, "Deleted contact at: " +
                insertUri.toString(), Toast.LENGTH_SHORT).show();
    updateUri = null;
    insertUri = null;
    Log.i(getClass().getSimpleName(),"Deleted the contact inserted by this
                                          program");
}
```

- Potrebne dozole u manifestu

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