Android Application Development, COMP 10073

Mohawk College, Winter 2021

Contacts Provider

https://developer.android.com/guide/topics/providers/contacts-provider

The Contacts Provider is a powerful and flexible Android component that manages the device's central repository of data about people, it includes:

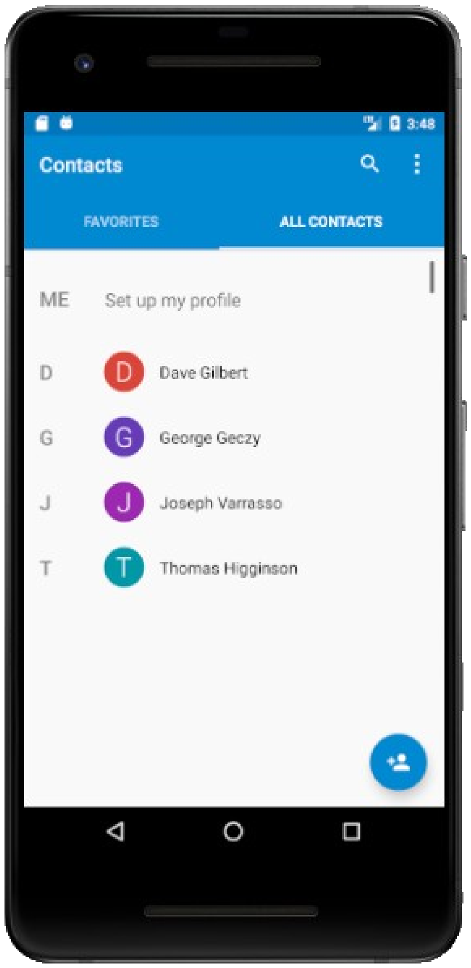
- Data you see in the device's contacts application
- Ability to access its data in your own application
- Ability to transfer data between the device and online services

The Contacts Provider maintains three types of data about a person, each of which corresponds to a table offered by the provider. The three tables are referred to by the names of their contract classes. The classes define constants for content URIs, column names, and column values used by the tables:

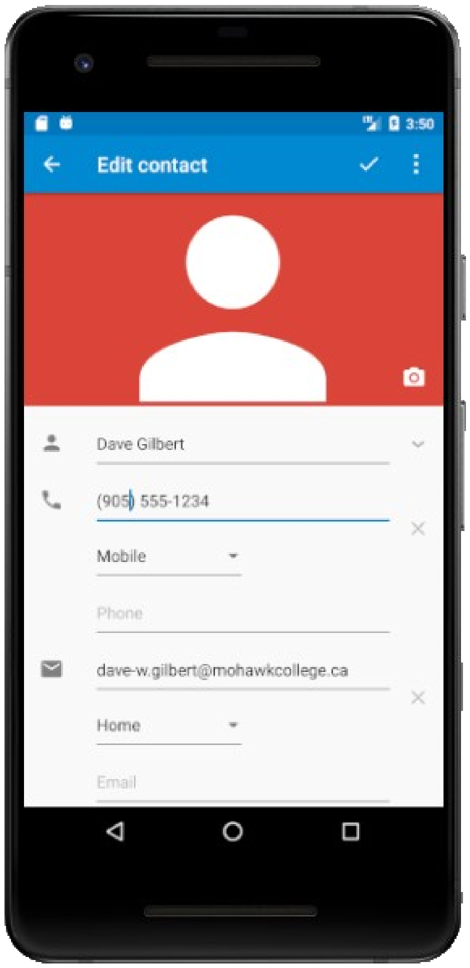
- ContactsContract.Contacts table
    - Rows representing different people, based on aggregations of raw contact rows.
- ContactsContract.RawContacts table
    - Rows containing a summary of a person's data, specific to a user account and type.
- ContactsContract.Data table
    - Rows containing the details for raw contact, such as email addresses or phone numbers.

Contacts Demo

For this demo to work you need to add some contacts to the emulator. Add 3 or 4 to get started:



D Dave Gilbert
G George Geczy
J Joseph Varrasso
T Thomas Higginson



Dave Gilbert
(905) 555-1234
Mobile
dave-w.gilbert@mohawkcollege.ca
Home

We need permissions, add to manifest.xml, & ask for them.

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

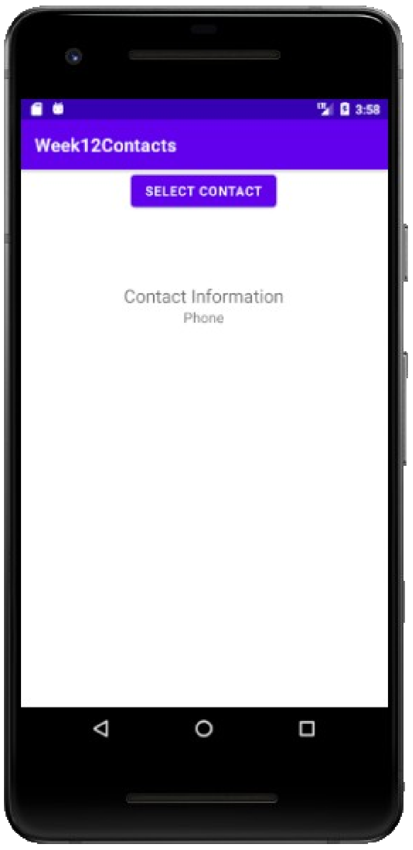
activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Contact Information"
        android:id="@+id/CName"
        android:textSize="20dp"
        android:textAlignment="center"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="128dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Phone"
        android:id="@+id/CPhone"
        android:textSize="16dp"
        android:layout_below="@+id/CName"
        android:textAlignment="center"
        android:layout_centerHorizontal="true" />

    <Button
        android:onClick="selectContact"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Select Contact"
        android:id="@+id/button"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```



Our activity uses a simple layout. We will select the name and the phone number from the internal contacts data base and display them for the user.

Our button will do dual duty, it will check to see if we have the contact permission, if we don't have it it will ask, if we do have it it will launch the contact picker through a special intent.

This isn't the best way to organize this code. Recall the GPS demo where we initiated an action after we were given permission. That is a better design, but for the purposes of a simple illustration this will do.

Add the onClick handler

```
/**
 * onClick Handler - start content pick activity, get permissions if necessary
 * @param view
 */
public void selectContact(View view) {
    // request permissions
    int hasWriteContactsPermission = ContextCompat.checkSelfPermission(this,
            Manifest.permission.READ_CONTACTS);
    if (hasWriteContactsPermission != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(this,
                new String[]{Manifest.permission.READ_CONTACTS}, ASK_CONTACT);
    } else {
        // Set up our ACTION_PICK Intent, and let Android start the pick activity
        Intent intent = new Intent(Intent.ACTION_PICK, ContactsContract.Contacts.CONTENT_URI);
        startActivityForResult(intent, PICK_CONTACT);
    }
    Log.d(TAG, "selectContact");
}
```

What is **Intent.ACTION_PICK?**

https://developer.android.com/reference/android/content/Intent#ACTION_PICK

Activity Action: Pick an item from the data, returning what was selected.

What is **ContactsContract.Contacts.CONTENT_URI**

https://developer.android.com/reference/android/provider/ContactsContract.Contacts.html

The MIME type of a CONTENT_URI subdirectory of a single person.

What is **startActivityForResult()**?

https://developer.android.com/training/basics/intents/result

Collects a result from the activity, compare to startActivity(). We used this in a previous lecture where the new activity that we started was able to return a result to the original activity. This particular usage. To take advantage of this we need to Override onActivityResult().

We define a method onActivityResult() to collect our contact info. Called when:

- an activity you launched exits
- giving you the requestCode you started it with
- the resultCode it returned
- any additional data from it.

The resultCode will be RESULT_CANCELED if the activity explicitly returned that, didn't return any result, or crashed during its operation. You will receive this call immediately before onResume() when your activity is re-starting.

MainActivity.java

```java
public class MainActivity extends AppCompatActivity {

    public static final String TAG = "==MainActivity==";
    public static final int PICK_CONTACT = 101;
    public static final int ASK_CONTACT = 1001;

    /**
     * onCreate (default)
     *
     * @param savedInstanceState - (default)
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(TAG, "onCreate");
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
        super.onActivityResult(requestCode, resultCode, intent);
        Log.d(TAG, "onActivityResult");
    }
}
```

Check for the PICK_CONTACT requestCode. Extract the URI from the intent. Use the URI as the root query to the ContentResolver to get a cursor. Extract the contactId and log the basic details to check that the app is working.

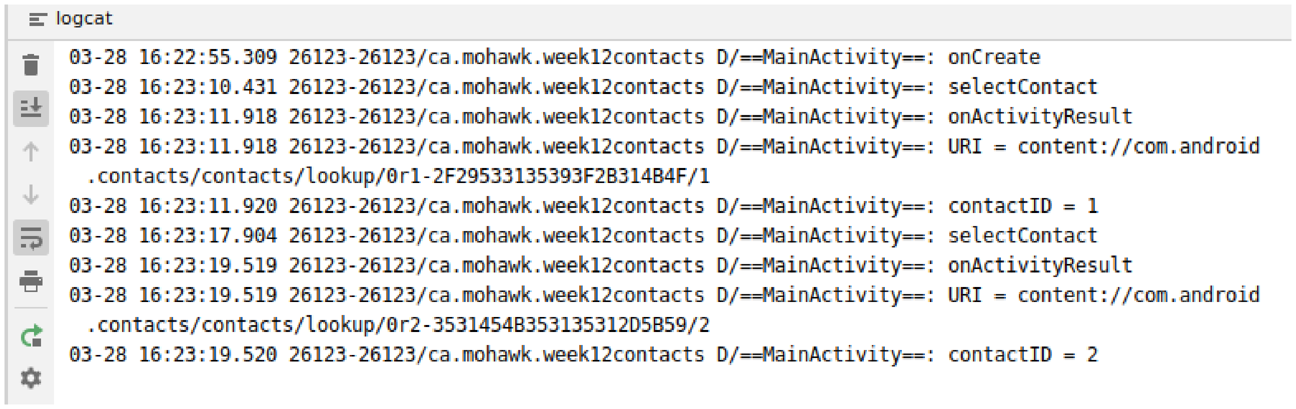
```java
if (requestCode == PICK_CONTACT) {
    if (resultCode == RESULT_OK) {
        boolean hasPhone = false;
        int numPhones = 0;
        String contactPhone = "";

        Uri uri = intent.getData();
        Log.d(TAG, "URI = "+ uri.toString());

        // Not using projection, return all columns....
        Cursor cursor = getContentResolver().query(uri, null,
                null,null, null);
        cursor.moveToFirst();

        // Store the contact ID to use with content resolver queries
        String contactId = cursor.getString(
                cursor.getColumnIndex(ContactsContract.Contacts._ID));
        Log.d(TAG, "contactID = " + contactId);
```

The Contacts URI



```
03-28 16:22:55.309 26123-26123/ca.mohawk.week12contacts D/==MainActivity==: onCreate
03-28 16:23:10.431 26123-26123/ca.mohawk.week12contacts D/==MainActivity==: selectContact
03-28 16:23:11.918 26123-26123/ca.mohawk.week12contacts D/==MainActivity==: onActivityResult
03-28 16:23:11.918 26123-26123/ca.mohawk.week12contacts D/==MainActivity==: URI = content://com.android
  .contacts/contacts/lookup/0r1-2F29533135393F2B314B4F/1
03-28 16:23:11.920 26123-26123/ca.mohawk.week12contacts D/==MainActivity==: contactID = 1
03-28 16:23:17.904 26123-26123/ca.mohawk.week12contacts D/==MainActivity==: selectContact
03-28 16:23:19.519 26123-26123/ca.mohawk.week12contacts D/==MainActivity==: onActivityResult
03-28 16:23:19.519 26123-26123/ca.mohawk.week12contacts D/==MainActivity==: URI = content://com.android
  .contacts/contacts/lookup/0r2-3531454B353135312D5B59/2
03-28 16:23:19.520 26123-26123/ca.mohawk.week12contacts D/==MainActivity==: contactID = 2
```

You should be able to extract some data from the log at this stage.

Notice that this isn't an http:// style URL, but rather a "content:// ... " URI. This reference is to a specific contacts database entry. We used the cursor to extract the column ID from the Column Index. Notice the familiar reference to **_ID**.

Accessing the details of the database requires some knowledge of the column names, but aside from this it is similar to reading the SQLite Database.

```
// Some data we can get right from the base data structure
// - display name, has phone number, photo
int nameColumnIndex = cursor.getColumnIndex(
        ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME);
String contactName = cursor.getString(nameColumnIndex);

// We could, if we wanted, run a query to retrieve a cursor to
// the Phone Numbers and count the records
// However, "HAS_PHONE_NUMBER" in the main contact
// record provides us with a true/false value,
int hasPhoneColumnIndex = cursor.getColumnIndex(
        ContactsContract.CommonDataKinds.Phone.HAS_PHONE_NUMBER);
// String will be "1" or "0"
hasPhone = cursor.getString(hasPhoneColumnIndex).equals("1");

if (hasPhone) {
    // You know it has a number so now query it like this
    Cursor phones = getContentResolver().query(
            ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null,
            ContactsContract.CommonDataKinds.Phone.CONTACT_ID +
                    " = "+ contactId, null, null);
    while (phones.moveToNext()) {
        numPhones++;
        contactPhone += phones.getString(phones.getColumnIndex(
                ContactsContract.CommonDataKinds.Phone.NUMBER)) + "\n";
    }
    phones.close();
}
Log.d(TAG, "Phone#s " + numPhones);
```

Update the TextView

Finally to make the application fully functional we need to update the TextView.

```
// Update the TextView
TextView nameText = (TextView) findViewById(R.id.CName);
nameText.setText(contactName);

TextView phoneText = (TextView) findViewById(R.id.CPhone);
phoneText.setText(contactPhone);
```

Add a Basic Notification

https://developer.android.com/training/notify-user/build-notification.html

https://developer.android.com/guide/topics/ui/notifiers/notifications.html

As an example of how not to use notifications, call this function from somewhere in your code. It will generate a low priority notification.

```
public void doNotify() {

    // Create an explicit intent for an Activity in your app
    Intent intent = new Intent(this, MainActivity.class);
    intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);

    // pendingIntent just restarts the current activity. Add it to the builder
    // with .setContentIntent(pendingIntent) so that we collect a result.
    // This version just restarts the MainActivity when you click on the star.
    PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);

    Log.d(TAG, "doNotify");

    NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
            .setSmallIcon(android.R.drawable.star_big_on)
            .setContentTitle("My notification - Low p")
            .setContentText("Hello World!")
            .setPriority(NotificationCompat.PRIORITY_MIN)
            .setContentIntent(pendingIntent);

    NotificationManagerCompat notificationManager = NotificationManagerCompat.from(this);

    // notificationId is a unique int for each notification that you must define
    notificationManager.notify(1001, builder.build());
}
```

Notifications cannot be used to fill out the breadth requirement for the project. A dialog fragment will make more sense in most instances. Only use notifications in your project if you have some sort of background activity, or a timer, or you want a special response to an external event outside the users control.