Android

Ben Gavan

August 3, 2019

Contents

1 Dialogs

1.1 Creating a Dialog View

A dialog is a type of fragment.

1.1.1 XML

To create the Dialog View, it is just like creating any other view.
Create a normal layout xml file and create the layout that is required for the
dialog.

Example: DatePicker :

```xml
<?xml version="1.0" encoding="utf-8"?>
<DatePicker xmlns:android="http://schemas.android.com/apk/res/
    android"
  android:id="@+id/dialog_date_picker"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:calendarViewShown="false">
</DatePicker>
```

1.1.2 The Class

A Dialog Fragment extends the class *DialogFragment*

```
1  public class DatePickerFragment extends DialogFragment {
2    ...
3  }
```

Example: DatePicker :

```
1
2  import android.app.Activity;
3  import android.app.Dialog;
4  import android.content.DialogInterface;
5  import android.content.Intent;
6  import android.os.Bundle;
7  import android.support.annotation.NonNull;
8  import android.support.v4.app.DialogFragment;
9  import android.support.v7.app.AlertDialog;
10 import android.view.LayoutInflater;
11 import android.view.View;
12 import android.widget.DatePicker;
13
14 import java.util.Calendar;
15 import java.util.Date;
16 import java.util.GregorianCalendar;
17
18 /**
19 * Created by ben on 26/10/2017.
20 */
21
22 public class DatePickerFragment extends DialogFragment {
23
24   public static final String EXTRA_DATE = "com.bgsoftwarestudios.
         criminalintent.date";
25
26   private static final String ARG_DATE = "date";
27
28   private DatePicker mDatePicker;
29
30   public static DatePickerFragment newInstance(Date date) {
31     Bundle args = new Bundle();
32     args.putSerializable(ARG_DATE, date);
33
34     DatePickerFragment fragment = new DatePickerFragment();
35     fragment.setArguments(args);
36     return fragment;
37   }
38
39   @NonNull
40   @Override
41   public Dialog onCreateDialog(Bundle savedInstanceState) {
42     Date date = (Date) getArguments().getSerializable(ARG_DATE);
43
44     Calendar calendar = Calendar.getInstance();
45     calendar.setTime(date);
46     int year = calendar.get(Calendar.YEAR);
```

```
47        int month = calendar.get(Calendar.MONTH);
48        int day = calendar.get(Calendar.DAY_OF_MONTH);
49
50        View view = LayoutInflater.from(getActivity()).inflate(R.layout
             .dialog_date, null);
51
52        mDatePicker = (DatePicker) view.findViewById(R.id.
             dialog_date_picker);
53        mDatePicker.init(year, month, day, null);
54
55        return new AlertDialog.Builder(getActivity())
56                 .setView(view)
57                 .setTitle(R.string.date_picker_title)
58                 .setPositiveButton(android.R.string.ok, new
                     DialogInterface.OnClickListener() {
59                   @Override
60                   public void onClick(DialogInterface dialogInterface,
                        int i) {
61                     int year = mDatePicker.getYear();
62                     int month = mDatePicker.getMonth();
63                     int day = mDatePicker.getDayOfMonth();
64                     Date date = new GregorianCalendar(year, month, day)
                          .getTime();
65                     sendResult(Activity.RESULT_OK, date);
66                   }
67                 })
68                 .create();
69    }
70
71    private void sendResult(int resultCode, Date date) {
72        if (getTargetFragment() == null) {
73          return;
74        }
75
76        Intent intent = new Intent();
77        intent.putExtra(EXTRA_DATE, date);
78
79        this.getTargetFragment().onActivityResult(this.
             getTargetRequestCode(), resultCode, intent);
80    }
81 }
```

1.2 Setting a Target Fragment

When displaying a dialog view from a fragment, we need to create a relationship between them se we can send data back from the dialog to the fragment.
We need to pass a reference to the dialog of the fragment, as well as a request code to identify the payload when it is sent back/ so the fragment can 'listen' out for it.
We do this by setting the target fragment on the dialog object:

```
1  dialog.setTargetFragment(FragmentClass.this, REQUEST_CODE);
```

1.3 Sending data back to the Target Fragment from the Dialog

We should also check that the target fragment has been set before we do anything

First, we need to get a reference to the target fragment (set by the fragment requesting the display of the dialog via using setTargetFragment on the dialog). We then call 'onActivityResult' on the target fragment.

So if we want to do something in the fragment i.e. get the data back, we have to override this method in the target fragment.

The data we pass back from the dialog is contained within an intent by putExtra.

```
1  private void sendResult(int resultCode, Date date) {
2    if (getTargetFragment() == null) {
3      return;
4    }
5
6    Intent intent = new Intent();
7    intent.putExtra(EXTRA_DATE, date);
8
9    this.getTargetFragment().onActivityResult(this.
         getTargetRequestCode(), resultCode, intent);
10 }
```

1.3.1 Receiving the Data from the intent

```
1  @Override
2  public void onActivityResult(int requestCode, int resultCode,
       Intent data) {
3    if (resultCode != Activity.RESULT_OK) {
4      return;
5    }
6
7    if (requestCode == REQUEST_DATE) {
8      Date date = (Date) data.getSerializableExtra(DatePickerFragment
           .EXTRA_DATE);
9      mCrime.setDate(date);
10     mDateButton.setText(mCrime.getDate().toString());
11   }
12 }
```

We override the 'onActivityResult' within the target fragment we are sending data back to.

First we check what the result code is (what button the user pressed on the dialog)

```
1  if (resultCode != Activity.RESULT_OK) {
2    return;
3  }
```

We then check what the request code is (which was set by the fragment creating the dialog) so we know that we are responding to the correct result (A fragment can display and react to multiple dialogs).

After this, we get the data sent back in the form of an extra from the dialog inside an intent by 'getSerializableExtra(...)'.

In this case, we cast this data back to a date so it can be used.

```
1  if (requestCode == REQUEST_DATE) {
2    Date date = (Date) data.getSerializableExtra(DatePickerFragment.
         EXTRA_DATE);
3    mCrime.setDate(date);
4    mDateButton.setText(mCrime.getDate().toString());
5  }
```

2 The Toolbar

The Toolbar provides additional mechanisms for navigation, nd also provides design consistency and branding.

2.0.1 History

The toolbar component was added to android 5.0 (Lollipop).
Prior to this, the action bar was the recommended component for navigation and actions within an app.
The toolbar and action bar are very similar.
The toolbar builds on top of the action bar .
It has a tweaked UI
It's more flexible in the ways you can use it.

2.0.2 Supported by

Since the toolbar has been added to the AppCompat library, it is available back to API 9 (Android 2.3)

2.1 Menus

The top-right portion of the toolbar is reserved for the toolbar's menu.
The menu consists of action items (sometimes referred to as menu items).
These can perform an action on the current screen or on the app as a whole.

2.1.1 Defining a menu in XML

Need to create an XML description of a menu, just like how you have to for layouts, with the resource file inside the res/menu directory.
To create a new menu resource file:

1. Right-click on the res directory

2. Select New → Android resource file

3. Change the Resource type to Menu

4. Name the resource (normally 'fragment_...' - the same naming convention as layout files)

5. Click OK

In this file, the XML should be:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <menu xmlns:android="http://schemas.android.com/apk/res/android"
3        xmlns:app="http://schemas.android.com/apk/res-auto">
4  </menu>
```

2.1.2 Defining an item

```
1  <item
2    android:id="@+id/new_crime"
3    android:icon="@android:drawable/ic_menu_add"
4    android:title="@string/new_crime"
5    app:showAsAction="ifRoom|withText"/>
```

The line

```
1  app:showAsAction="ifRoom|withText"
```

makes the item be displayed inline/on the toolbar (where the menu icon should be) instead of having the item as a drop down item below the toolbar/menu button.
The showAsAction attribute refers to whether the item will appear in the toolbar itself or in the overflow menu.

In this case "ifRoom|withText" will make the items icon and text appear in the toolbar if there is room.
If there is room for the icon but not the text, then only the icon will be visible.
If there is no room for either, the item will be relegated to the overflow menu.

If there are items in the overflow menu, the three dots will appear and when these are pressed, the overflow menu will be shown below.

Multiple menu items can be displayed as Actions on the Toolbar.

Possible values for showAsAction

- always

    not recommended

    Better to use ifRoom and let the OS decide.

- ifRoom

    Only displays the item as an Action if there is room

- never

    never displayed as an action

will always appear in the overflow menu

so good for items that are not used very often - its good practice to avoid having to many items on the toolbar to help the screen keep decluttered

The AppCompat library defines its own custom showAsAction attribute and does not look for the native showAsAction attribute.

2.1.3 Creating the Menu

Override the function onCreateOptionsMenu(...) inside the Activity/Fragment. To actually create/inflate the menu: (Inside the Fragment:)

```
1  @Override
2  public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
3    super.onCreateOptionsMenu(menu, inflater);
4    inflater.inflate(R.menu.fragment_crime_list, menu);
5  }
```

This populates the menu with the items defined in the menu/fragment_crime_list.xml file.
The super call is only convention since the superclass, Fragment, does nothing. (Good to do so the superclass functionality is still applied - can now change the superclass and will still work if we do something in that implementation of this function).

We then need to call setHasOptionsMenu(boolean hasMenu) to tell the FragmentManager that this fragment has a menu and should receive a call to onCreateOptionsMenu(...).
Inside the Fragment:

```
1  @Override
2  public void onCreate(@Nullable Bundle savedInstanceState) {
3    super.onCreate(savedInstanceState);
4    setHasOptionsMenu(true);
5  }
```

2.1.4 Responding to Menu Selection

Override onOptionsItemSected(...) in the fragment that you have called 'setHasOptionsMenu(true)'.
The MenuItem.getItemId() corresponds to the id of the <item> which you set in the xml file for the menu.
This means that we can perform a switch case for each possible id in the menu. Include a default case to let the super implementation to handle the section of any item that you have not declared.
You should return true you you have handled the item section and that no further processing is necessary.

```
1  @Override
2  public boolean onOptionsItemSelected(MenuItem item) {
3    switch (item.getItemId()) {
4      case R.id.new_crime:
5        .... Do some logic here ....
6        return true;   // Return true to say that the selection has
               been handled.
7      default:
8        return super.onOptionsItemSelected(item);
9    }
10 }
```

2.1.5 Reload/update the menu

```
1  getActivity().invalidateOptionsMenu();
```

This will cause the menu to be redrawn/reloaded (just like if the device is rotated).

2.2 Subtitle

```
1  private void setSubtitle(String subtitle) {
2    AppCompatActivity activity = (AppCompatActivity) getActivity();
3    activity.getSupportActionBar().setSubtitle(subtitle);
4  }
```

- Get the current activity

    (We are using AppCompat for backwards compatibility)

- Get the Toolbar from that activity via getSupportActionBar()

    Still called/referred to as an Action Bar due to legacy reasons.

- Set the subtitle of that the toolbar we just received.

2.3 Hierarchical Navigation

Add parentActivityName to the activity the the manifests so when you press the back arrow on the toolbar, it will go back to the activity you stated.

```
1  <activity
2    android:name=".CrimePagerActivity"
3    android:parentActivityName=".CrimeListActivity"/>
```

2.3.1 How Hierarchical Navigation works

Page 261

10

3 AppCompat Library

3.1 Requirements

The AppCompat requires that you:

- add the AppCompat dependency

- use one of the AppCompat themes

- ensure that all activities are a subclass of AppCompatActivity

4 SQLite Database

4.1 Defining a Schema

```
1  public class CrimeDbSchema {
2
3    public static final class CrimeTable {
4      public static final String NAME = "crimes";
5
6      public static final class Cols {
7        public static final String UUID = "uuid";
8        public static final String TITLE = "title";
9        public static final String DATA = "date";
10       public static final String SOLVED = "solved";
11     }
12   }
13 }
```

4.2 Building the Initial Database

Always need to follow a few basic steps:

- Check to see whether the database already exists.

- If it does not, create it and create the tables and initial data it needs.

- If id does, open it and see what version of the schema it has.

- If it is an old version, upgrade it to a newer version.

SQLiteOpenHelper can be used to handle all of this.

4.3 Opening an SQLiteDatabase

By extending SQLiteOpenHelper, we give control over to it to do the heavy lifting in opening the database.

```
1  public class CrimeBaseHelper extends SQLiteOpenHelper {
2
3    private static final int VERSION = 1;
4    private static final String DATABASE_NAME = "crimeBase.db";
```

```
 5
 6    public CrimeBaseHelper(@Nullable Context context) {
 7      super(context, DATABASE_NAME, null, VERSION);
 8    }
 9
10    @Override
11    public void onCreate(SQLiteDatabase sqLiteDatabase) {
12
13    }
14
15    @Override
16    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int
          i1) {
17
18    }
19 }
```

To access the database we can then call getWritableDatabase()

```
1  private CrimeLab(Context context) {
2    this.mContext = context.getApplicationContext();
3    this.mDatabase = new CrimeBaseHelper(mContext).
          getWritableDatabase();
4  }
```

When we do this, SQLiteOpenHelper will:

- open up /data/data/com....../databases/thedatabasebeingopened.db

  it will create a new database file if it does not already exist.

- If it is the first time the database has been created, call onCreate(...), then save out the latest version number.

- If it is not the first time, check the version number.

  If the version number in CrimeBaseHelper is higher, call onUpgrade(....)

4.4 Writing to the Database

4.4.1 ContentValues

Writes and updates are done with ContentVales - a key-value store class, like Java's HashMap or Bundles.
Example of a helper function to create the instance of ContentValues for a row:

```
1  private static ContentValues getContentValues(@NonNull Crime crime)
          {
2    ContentValues values = new ContentValues();
3    values.put(CrimeTable.Cols.UUID, crime.getId().toString());
4    values.put(CrimeTable.Cols.TITLE, crime.getTitle());
5    values.put(CrimeTable.Cols.DATE, crime.getDate().getTime());
6    values.put(CrimeTable.Cols.SOLVED, crime.isSolved() ? 1 : 0);
7
8    return values;
9  }
```

4.4.2 Inserting rows

Can insert a new row to the database by using the content values object, and using the insert(...,...,...) method on the SQLite database object.

```
1  public void addCrime ( Crime crime ) {
2     ContentValues  values = getContentValues ( crime );
3     mDatabase . insert ( CrimeTable . NAME , null , values );
4  }
```

4.4.3 Updating Rows

```
1  public void updateCrime ( Crime crime ) {
2     String uuidString = crime . getId (). toString ();
3     ContentValues  values = getContentValues ( crime );
4     mDatabase . update ( CrimeTable . NAME , values ,
5        CrimeTable . Cols . UUID + " = ?" ,
6        new String [] { uuidString });
7  }
```

To update a row, the same content values object is used from inserting; however, the update(...,...,...,...) method is called in the database object.

The third parameter is the where clause string which specifies what rows are updated. In this case, the UUID is used to identify the row.

To do this, the '?' syntax is used which tells the database to treat whatever string is in the following parameter as a pure string - not as SQL code. This prevents an SQL injection attack.

4.5 Reading from the Database

Reading from the database is done by using the query(...) function.
This returns a 'Cursor' object.
A cursor stores the retrieved data in key value pairs.

4.5.1 Retrieving a Cursor

```
1  public Cursor queryCrimes ( String whereClause , String [] whereArgs ) {
2     Cursor cursor  = mDatabase . query (
3        CrimeTable . NAME ,
4        null ,  // selects all columns
5        whereClause ,
6        whereArgs ,
7        null , // groupBy
8        null ,  // having
9        null  // orderBy
10    );
11    return cursor ;
12 }
```

4.5.2 Using a Cursor

To actually retrieve the returned data/values, the get[Type]([Int]) function is used, where the Int is the key with the value of the column index, and the Type

is the type of value which is stored.

To get the column index from the column name/title, the getColumnIndex([String]) can be used.

```
1  String title = getString(getColumnIndex(CrimeTable.Cols.TITLE));
2  long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
3  int isSolved = getInt(getColumnIndex(CrimeTable.Cols.SOLVED));
```

It is cleaning, however, to use a custom wrapper of a cursor to encapsulate the cursor and retrieving of data withing one object.

Therefore create a class which extends Cursor

```
1  public class CrimeCursorWrapper extends CursorWrapper {
2
3    public CrimeCursorWrapper(Cursor cursor) {
4      super(cursor);
5    }
6
7    public Crime getCrime() {
8      String uuidString = this.getString(this.getColumnIndex(
          CrimeTable.Cols.UUID));
9      String title = getString(getColumnIndex(CrimeTable.Cols.TITLE))
          ;
10     long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
11     int isSolved = getInt(getColumnIndex(CrimeTable.Cols.SOLVED));
12
13     Crime crime = new Crime(UUID.fromString(uuidString));
14     crime.setTitle(title);
15     crime.setDate(new Date(date));
16     crime.setSolved(isSolved != 0);
17
18     return crime;
19   }
20 }
```

From this point, convert the retrieved data into model objects.

To move the cursor along from one part of the query to the next, use the Cursor.moveToFirst() to move to the beginning of the query and Cursor.moveToNext() to move to the next position.

To check is the cursor is still inside the data set, using Cursor.isAfterLast()

Hence the name, cursor.

```
1  public List<Crime> getCrimes() {
2    List<Crime> crimes = new ArrayList<>();
3    CrimeCursorWrapper cursorWrapper = queryCrimes(null, null);
4
5    try {
6      cursorWrapper.moveToFirst();
7      while (!cursorWrapper.isAfterLast()) {
8        crimes.add(cursorWrapper.getCrime());
9        cursorWrapper.moveToNext();
10     }
11   } finally {
12     cursorWrapper.close();
13   }
14
15   return crimes;
```

```
16  }
```

Remember to close the cursor.

If you don't the app will run out of open file handlers and the app will crash.

Example of retrieving specific row:

```
1   public Crime getCrime(UUID id) {
2     CrimeCursorWrapper cursor = queryCrimes(
3       CrimeTable.Cols.UUID + "",
4       new String[] { id.toString() }
5     );
6
7     try {
8       if (cursor.getCount() == 0) {
9       return null;
10      }
11
12      cursor.moveToFirst();
13      return cursor.getCrime();
14      } finally {
15        cursor.close();
16      }
17  }
```

4.6   Deleting Rows

```
1   mDatabase.delete(CrimeTable.NAME,
2             CrimeTable.Cols.UUID + " = ?",
3             new String[] { crime.getId().toString() });
```

5   Implicit Intents

Implicit intents are used to start activities in another app.

In an implicit intent, you describe the job you require to be completed, and the OS will open an appropriate activity.

Compared to Explicit intents where you specify the class of the activity to start.

5.1   The Parts of an Implicit Intent

- action

  Typically constants from the Intent class.

- location of any data

- type of data that the action is for

- optional categories

They can also include extras. However, there are not used by the OS to find the most appropait

5.2 Advertising an Activity to Accept Implicit Intents

For example, to advertise an activity's capability to handle an implicit intent to open a web page, the following has the be declared within the AppManifest file.

```
1  <activity android:name=".BrowserActivity">
2    <intent-filter>
3      <action android:name="android.intent.action.VIEW"/>
4      <category android:name="android.intent.category.DEFAULT"/>
5      <data android:scheme="http" android:host="www.somewhere.com"/>
6    </intent-filter>
7  </activity>
```

5.3 Sending Text

5.3.1 With option of send

```
1  Intent intent = new Intent(Intent.ACTION_SEND);
2  intent.setType("text/plain");
3  intent.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
4  intent.putExtra(Intent.EXTRA_SUBJECT, getString(R.string.
       crime_report_subject));
5  intent = Intent.createChooser(intent, getString(R.string.
       send_report));
6  startActivity(intent);
```

5.4 Requesting Android for a Contact

5.4.1 The Request of Data

The implicit intent action will be Intent.ACTION_PICK.
Since we are expecting data to be sent back, the activity will be started with $startActivityForResult(...)$ along with a request code to be able to identify the sent back response/result.

```
1  final Intent pickContactIntent = new Intent(Intent.ACTION_PICK,
       ContactsContract.Contacts.CONTENT_URI);
2  startActivityForResult(pickContactIntent, REQUST_CONTACT);
```

5.4.2 The Receival of Data

Inside $onActivityResult(...,...,...)$

```
1  if (data == null) {
2    break;
3  }
4
5  Uri contactUri = data.getData();
6  String[] queryFields = new String[] {
7    ContactsContract.Contacts.DISPLAY_NAME
8  };
9  Cursor cursor = getActivity().getContentResolver().query(contactUri
       , queryFields, null, null, null);
```

```
10  try {
11    if (cursor.getCount() == 0) {
12      return;
13    }
14
15    cursor.moveToFirst();
16    String suspect = cursor.getString(0);
17    mCrime.setSuspect(suspect);
18    mSuspectButton.setText(suspect);
19  } finally {
20      cursor.close();
21  }
```

5.4.3   Checking if the Device has a Contacts App

Use the OS Package manager to check if the device has a contacts app.
If it does not, and you request data from the contacts app, the app will crash.
It is therefore recommended to deactivate the functionality that uses requires
this.

```
1  PackageManager packageManager = getActivity().getPackageManager();
2  if (packageManager.resolveActivity(pickContactIntent ,
       PackageManager.MATCH_DEFAULT_ONLY) == null) {
3    mSuspectButton.setEnabled(false);
4  }
```

This request returns an instance of *ResolveInfo* telling all about what the
activity it found.

5.5   Taking Pictures with intents

All things Media related is defined in *MediaStore*. It defines the public inter-
faces used in Android for interacting with common media.
The camera intent is defined in here as *MediaStore.ACTION_IMAGE_CAPTURE*.

By default *ACTION_IMAGE_CAPTURE* will take a thumb-nail picture and
return it inside the *Intent* object returned in *onActivityResult(...)*
For a full-resolution picture, you need to tell it where to store the file on the file
system.
This can be completed by passing a Uri pointing to where you want to save
the file in *MediaStore.EXTRA_OUTPUT*. This Uri will point to a location
serviced by *FileProvider*.

```
1  PackageManager packageManager = getActivity().getPackageManager();
2  mPhotoButton = (ImageButton) view.findViewById(R.id.crime_camera);
3  final Intent captureImageIntent = new Intent(MediaStore.
       ACTION_IMAGE_CAPTURE);
4  boolean canTakePhoto = (mPhotoFile != null) && (captureImageIntent.
       resolveActivity(packageManager) != null);
5  mPhotoButton.setEnabled(canTakePhoto);
6  mPhotoButton.setOnClickListener(new View.OnClickListener() {
7    @Override
8    public void onClick(View view) {
```

```
 9      Uri uri = FileProvider.getUriForFile(getActivity(), "",
            mPhotoFile);
10      captureImageIntent.putExtra(MediaStore.EXTRA_OUTPUT, uri);
11
12      List<ResolveInfo> cameraActivities = getActivity().
            getPackageManager().queryIntentActivities(
            captureImageIntent, PackageManager.MATCH_DEFAULT_ONLY);
13
14      for (ResolveInfo activity : cameraActivities) {
15        getActivity().grantUriPermission(activity.activityInfo.
              packageName, uri, Intent.FLAG_GRANT_WRITE_URI_PERMISSION)
              ;
16      }
17
18      startActivityForResult(captureImageIntent, REQUEST_PHOTO);
19    }
20 });
```

5.5.1   Declaring the Camera Feature

To declare that the app uses the camera, add the following into $AndroidManifest.xml$:

```
1 <uses-feature android:name="android.hardware.camera"
2       android:required="false"/>
```

6   File Storage

Can store files in out own private storage for the app. This is the same location where the SQLite database is stored. So your app is the only one able to access them.

These files are access using the *Context* class.

The primary methods in the *Context* class:

- *getFilesDir()*

To let other apps access files stored in this location, we can use a *ContentProvider*. This allows us to expose content URIs to other apps; which intern, allows us those apps to read and write to that specific URI.

6.1   FileProvider

When all that is required is to be able to receive a file from another application, a *FileProvider* can be used instead of implementing an entire *ContentProvider*; which in this case would be classed as overkill.

The first step is to declare a *FileProvider* as a *ContentProvider* hooked up to a specific authority. This is done by adding a content provider declaration to $AndroidManifest.xml$

The authority is a location where files can be saved to.

This gives other apps a target to write to.

```
1  <provider
2    android:authorities="com.bgsoftwarestudios.criminalintent.
        fileprovider"
3    android:name="android.support.v4.content.FileProvider"
4    android:exported="false"
5    android:grantUriPermissions="true" />
```

The $android : exported = "false"$ attribute stops any app from using this
provider except you and apps that you grant permission to.
And the $android : grantUriPermissions = "true"$ attribute allows us to give
other apps permission to write to this URI when we send out an intent.

6.1.1 Exposing/Telling the *FileProvider* what files it is exposing

To tell the *FileProvider* what files to expose, create a new resource file $xml/files.xml$.
You can create this initially by:

1. Right-click on the app/res directory.

2. Select new → Android resource file.

3. For Resource type select XML.

4. Enter $files$ for the file name.

5. Enter the flowing into the the xml file

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <paths>
3    <files-path
4      name="crime_photos"
5      path="."/>
6  </paths>
```

This declares the file paths that *FileProvider* will use/expose internally.
We then need to hook up the *files.xml* to the *FileProvider* by using a $meta-data$ tag in $AndroidManifest.xml$.

6.1.2 Hooking up the paths description to the *FileProvider* within $AndroidManifest.xml$

Add a $meta-data$ element inside the provider:

```
1  <provider
2    android:authorities="com.bgsoftwarestudios.criminalintent.
        fileprovider"
3    android:name="android.support.v4.content.FileProvider"
4    android:exported="false"
5    android:grantUriPermissions="true">
6    <meta-data
7      android:name="android.support.FILE_PROVIDER_PATHS"
8      android:resource="@xml/files"/>
9  </provider>
```

6.1.3 Revoking File write *FileProvider* permissions

When the picture is taken, the method *onActivityResult*(..., ..., ...) is called. So when the request comes back we can remove the file write permissions to that Uri for that external activity.

```
1  Uri uri = FileProvider.getUriForFile(getActivity(),
       FILE_PROVIDER_AUTHORITY, mPhotoFile);
2  getActivity().revokeUriPermission(uri, Intent.
       FLAG_GRANT_WRITE_URI_PERMISSION);
3  updatePhotoView();
```

7   Bitmaps

Bitmaps store image data as literal pixel data

7.1   Scaling Bitmaps

A 16-megapixel, 24-bit camera image compressed as a JPG with a size of 5 MB, would be 48 MB as a bitmap.
To shrink a bit map:

1. scan the file to determine the size

2. figure out how much it needs to be scaled for the desired dimensions.

3. reread the file and construct the new bitmap.

```
1  public class PictureUtils {
2
3    public static Bitmap getScaledBitmap(String path, int destWidth,
         int destHeight) {
4      // Read in the dimentions of the image on disk
5      BitmapFactory.Options options = new BitmapFactory.Options();
6      options.inJustDecodeBounds = true;
7      BitmapFactory.decodeFile(path, options);
8
9      float srcWidth = options.outWidth;
10     float srcHeight = options.outHeight;
11
12     // Calculate how much to scale down by
13     int inSampleSize = 1;
14     if (srcHeight > destHeight || srcWidth > destWidth) {
15       float heightScale = srcHeight / destHeight;
16       float widthScale = srcWidth / destWidth;
17       float scale = heightScale > widthScale ? heightScale :
             widthScale;
18       inSampleSize = Math.round(scale);
19     }
20
21     options = new BitmapFactory.Options();
22     options.inSampleSize = inSampleSize;
23
```

```
24        // Read in and create final bitmap
25        return BitmapFactory.decodeFile(path, options);
26    }
27 }
```

The *inSampleSize* determines how big each new pixel is relative to each old pixel.

E.g. for a sample size of 2, one new pixel horizontally is equivalent to 2 old pixels horizontally; therefore, shrinking the bitmap horizontally by 2 (so the overall size will the one fourth).

7.1.1   Very conservative scaling

A very conservative scaling of bit map is to shrink it down to the size of an activity. This ensures that the image will never be too small for the size of the activity; therefore, keeping a high enough quality/resolution.

```
1 public static Bitmap getScaledBitmap(String path, @NonNull Activity
        activity) {
2    Point size = new Point();
3    activity.getWindowManager().getDefaultDisplay().getSize(size);
4    return getScaledBitmap(path, size.x, size.y);
5 }
```

8   Strings

8.1   Plurals

```
1 <plurals name="subtitle_plural">
2    <item quantity="one">%1$d crime</item>
3    <item quantity="other">%1$d crimes</item>
4 </plurals>
```

To retrieve/use the string:

```
1 getResources().getQuantityString(R.plurals.subtitle_plural,
       crimeSize, crimeSize);
```

8.2   Percentages

```
1 <string name="playback_speed_percentage">PlayBack Speed: %d%%</
       string>
```

9   Two-Pane Master-Detail

This is most commonly used for tablets.
However, you swill commonly require different layout for on phone vs tablet.
To do this, use a alias resource.

9.1 Alias Resource

In *res/values*, create a resource file for the default (phone) and list the layout to be used. Call this file *refs.xml*.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3    <item name="activity_masterdetail" type="layout">@layout/
         activity_fragment</item>
4  </resources>
```

Then for the layouts that are needed to be used for larger screen sizes include the Smallest Screen Width qualifier with a value of 600.
Call this file *refs.xml* also.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3    <item name="activity_masterdetail" type="layout">@layout/
         activity_twopane</item>
4  </resources>
```

This will display the later when the minimum screen dimension is 600dp.
To use this in code, refer to the layout you want using the name of the item.
The file/layout that is then displayed/used is declared as the string in the *item* element.

9.2 Determining Device size

page 335.

10 Localization

See chapter 18

11 Accessibility

11.1 TalkBack

TalkBack if an Android screen reader made by Google.

See Chapter 19.

12 Styles and Themes

12.1 Styles

Declared inside the *styles.xml* file

```
1  <style name="BeatBoxButton">
2    <item name="android:background">@color/dark_blue</item>
3  </style>
```

12.2 Style Inheritance

The second style declaration of *BeatBoxButton.Strong* will inherit all of the items declared in the style declared above it - *BeatBoxButton*.

```
1  <style name="BeatBoxButton">
2    <item name="android:background">@color/dark_blue</item>
3  </style>
4
5  <style name="BeatBoxButton.Strong">
6    <item name="android:textStyle">bold</item>
7  </style>
```

13 Assets

13.1 Why Assets over Dependencies

More basic/less overhead.
For sound files, we can store them in the 'res/raw' folder withing
The resource system is limited to a flat hierarchy, unlike assets which can implement its own custom file structure. This therefore is more organized when there are large amounts of assets.
Resources do not allow you to read in multiple files at once - you have to refer to each file independently (unlike assets where we can get the file list for a folder and then loop through each asset, retrieving each one at a time). Resources are given ids such as R.raw.file.

13.2 Creating an Assets Folder

To create an assets folder for your app:

- Go to the Android option for the folder/files layout

- Right-click → New → Folder (Last section of the menu with the android guy as the symbol to the left of the options)→ Assets Folder.

- Keep 'Change Folder Location' unchecked.

- select 'main' for 'Target Source Set'

- Press finish.

- Proceed to create sub directories for organization of your assets.

13.3 Accessing Assets

Assets are accessed using the *AssetsManager* class.

13.3.1 Getting an *AssetsManager*

You get get at *AssetsManager* from any context.

13.3.2 Getting assets files names

```
1  private void loadSounds() {
2    String[] soundNames;
3    try {
4      soundNames = assetManager.list(SOUNDS_FOLDER);
5      Log.i(TAG, "Found " + soundNames.length + " sounds");
6    } catch (IOException ioe) {
7      Log.e(TAG, "Could not list assets", ioe);
8      return;
9    }
10 }
```

where $SOUNDS\_FOLDER$ the directory within the Assets folder that you want to access.

13.4 SoundPool

SoundPool can load lots of sound files into memory and control the maximum number of sounds that are playing back at one time.

A benefit of *SoundPool* over other methods of playing sounds is that you you ask it to play, there is very little lag and starts to play almost immediately.

A trade off is that you are required to load the sound before it is played.

13.4.1 Creating a SoundPool

The *SoundPool* constructor takes the maximum number of sounds that can be played at any one time, the type of *AudioManager* you require (for music use $STREAM_{M}USIC$), and the third is the sample rate converter (the documentation says it is ignored).

```
1  soundPool = new SoundPool(MAX_SOUNDS, AudioManager.STREAM_MUSIC, 0)
       ;
```

If you are playing the maximum number of sounds and then you try to play another, the oldest sound will be stopped to make space for the new sound being added.

The AudoManager.*** also specifies what audio volume is adjusted when it is playing (specifies what volume is used to play that sound).

13.4.2 Loading Sounds

Give each sound a unique ID - typically an integer value. Use *Integer* so that it can have an unspecified value of *null*.

```
1  private void load(@NotNull Sound sound) throws IOException {
2    AssetFileDescriptor assetFileDescriptor = assetManager.openFd(
         sound.getAssetPath());
3    int soundId = soundPool.load(assetFileDescriptor, 1); // Loads a
         file into soundPool for later play back.
4    sound.setSoundId(soundId);
5  }
```

soundPool.load(...) loads a file into *soundPool* for later playback. It also returns an ID to keep track of it so it is able to play it or unload it at a later time.

assetManager.openFd(...) throws the *IOException*.

13.4.3 Playing Sounds

```
1   soundPool.play(soundId, 1f, 1f, 1, 0, 1f);
```

'priority' is ignored.
For 'loop', '-1' causes it to loop forever. 0 = do not loop.

14 Unit Testing

14.1 Creating a Test Class

1. Go to the class you want to test.

2. 'command + shift + T' to navigate to the test class

3. if there is no test class, select 'Create New Test...'

4. Select 'JUnit4'

5. set 'setUp/@Before' checked

6. Keep everything else unchecked.

7. Press OK.

8. Choose between 'androidTest' and 'test'

14.1.1 *androidTest* vs *test*

andoridTests   are run at run-time on a device or emulator withing the android environment.

tests   are run the the development machine outside of the android environment.

*test*s tend to be faster

14.2 Setting Up the Test

14.2.1 Using Mocked Dependencies

14.3 Writing Tests

15 Refactoring Techniques and Tools

There are many techniques and tools that can be used to make refactoring code easier.

15.1 Extracting a method with Android Studio

1. Highlight the code that you want to be extracted

2. Right-click and select Refactor $\rightarrow$ Extract $\rightarrow$ Method

3. Set the Visibility and method Name

4. Press Refactor or Preview to preview the changes

5. If there are multiple occurrences of the highlighted text being extracted, android studio will ask if you want to replace these as well

   You can either replace each occurrence one by one, or by choosing all.