

Android

Ben Gavan

July 18, 2019

Contents

1	Dialogs	2
1.1	Setting a Target Fragment	2
1.2	Sending data back to the Target Fragment from the Dialog . . .	2
1.2.1	Receiving the Data from the intent	2
2	The Toolbar	3
2.0.1	History	3
2.0.2	Supported by	3
2.1	Menus	3
2.1.1	Defining a menu in XML	4
2.1.2	Defining an item	4
2.1.3	Creating the Menu	5
2.1.4	Responding to Menu Selection	6
2.1.5	Reload/update the menu	6
2.2	Subtitle	6
2.3	Hierarchical Navigation	6
2.3.1	How Hierarchical Navigation works	7
3	AppCompat Library	7
3.1	Requirements	7
4	SQLite Database	7
4.1	Defining a Schema	7
4.2	Building the Initial Database	7
4.3	Opening an SQLiteDatabase	8
4.4	Writing to the Database	8
5	Refactoring Techniques and Tools	9
5.1	Extracting a method with Android Studio	9

1 Dialogs

1.1 Setting a Target Fragment

When displaying a dialog view from a fragment, we need to create a relationship between them so 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);
```

1.2 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.  
10         getTargetRequestCode(), resultCode, intent);  
11 }
```

1.2.1 Receiving the Data from the intent

```
1 @Override  
2 public void onActivityResult(int requestCode, int resultCode,  
3     Intent data) {  
4     if (resultCode != Activity.RESULT_OK) {  
5         return;  
6     }  
7  
8     if (requestCode == REQUEST_DATE) {  
9         Date date = (Date) data.getSerializableExtra(DatePickerFragment.  
10             EXTRA_DATE);  
11         mCrime.setDate(date);  
12         mDateButton.setText(mCrime.getDate().toString());  
13     }  
14 }
```

```

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, and 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 AppCompatActivity 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 too many items on the toolbar to help the screen keep decluttered

The AppCompatActivity 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 `onOptionsItemSelected(...)` 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.supportActionBar().setSubtitle(subtitle);
4 }
```

- Get the current activity
(We are using AppCompatActivity 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

3 AppCompatActivity Library

3.1 Requirements

The AppCompatActivity requires that you:

- add the AppCompatActivity dependency
- use one of the AppCompatActivity 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 it 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
17                          i1) {
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).
4         getWritableDatabase();
5 }
```

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     {
3         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 and updating rows

5 Refactoring Techniques and Tools

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

5.1 Extracting a method with Android Studio

1. Highlight the code that you want to be extracted
2. Right-click and select Refactor → Extract → 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.