Worksheet 5: App Interaction

Updated: 6th October, 2018

In this worksheet, you'll practice implementing the app interaction concepts covered in the lecture. For this purpose, you will create a very basic "test bed" system, or systems (if you prefer to divide things up). The structural aspects of app design have already been covered in previous worksheets, so there's no need to labour the point here.

Your app(s) here will have no functionality other than (a) very basic user input, (b) contacting other apps and retrieving certain data, and (c) displaying that data. You can have a single activity, with or without a fragment. If you choose *not* to have a fragment, be aware that the lecture notes assume otherwise, and (at various points) show a call to `getActivity()`.

Other Apps on the Emulator

So far we've treated the Android emulator as just a place to run our app, but (at a software level) it actually represents an entire fully-functional Android device. It comes loaded with a range of existing apps, including (most importantly) a call app, a contacts app, a camera app, and a maps app.

The call app requires no particular setting up. It isn't going to make real phone calls[1], but it can simulate outgoing (and incoming) calls.

The camera app requires no particular setting up either, although you may wonder *what*, exactly, you can possibly take a photo of. As you will discover, the emulator's "camera" is connected to a "virtual scene". Here you can use the mouse and keyboard to move and rotate the virtual camera.

(See developers.google.com/ar/develop/java/emulator#control_the_virtual_scene for details.)

The contacts app *does* require some setting up. Specifically, you need to enter the details of, say, two or more (fake) people – names, emails and phone numbers should suffice. The app will also ask you to "Add Account", but this is unnecessary and can simply be skipped.

The maps app too will ask you to "Sign in", but again this is unnecessary.

Note: The map app may not respond well to the mouse scroll wheel. In case you'd like to zoom or rotate the map, which usually requires a two-finger "pinch" gesture, you can hold down Ctrl (or Command on MacOS) and the left mouse button, and move the mouse.

[1] You can't be connected to the phone network without a SIM card.

1. Dialing Numbers and Showing Map Locations

Create a simple UI that will allow the user to enter a phone number, and a latitude and longitude, and include "Call" and "Show Location" buttons.

When pressed:

(a) The "Call" button should launch the call app, with the entered phone number. (You can experiment with both `ACTION_DIAL` and `ACTION_CALL` here.)

(b) The "Show Location" button should launch the maps app with the specified co-ordinates.

2. Capturing a Thumbnail Photo

Implement a button to launch the camera app to take a small ("thumbnail") photo. When the photo is returned back to your app, display it in an `ImageView` element.

Remember that you will need to override `onActivityResult()` in order to receive the photo.

> **Note:** In case you're using a fragment, there is indeed an overridable `onActivityResult()` method in the fragment as well as the activity, and it works the same way.
>
> The OS only knows about activities, so `Activity.onActivityResult()` is called first, but this then calls `Fragment.onActivityResult()` for each fragment.

3. Obtaining Contact Information

Implement a button to select a contact, and include fields to store the contact's ID, name, email and phone number.

> **Note:** We wouldn't normally show the ID to the user, but it may be helpful for *you* to see it.

This is more complicated, so it's best to attempt it in three steps:

(a) Obtain the ID and name.

You will need to implement the query in `onActivityResult()`. If you're working in the same test bed app that implements thumbnail photos, you will need to define and use non-conflicting request codes.

(b) Obtain the email address. The lecture notes show how to do this. Don't forget to modify `AndroidManifest.xml` to give your app the extra permission it needs.

(c) Obtain the phone number. This isn't directly shown in the lecture notes, but is *almost* identical to obtaining the email address.

> **Note:** The schema class is `ContactsContract.CommonDataKinds.Phone`. Look up the online reference documentation, and observe the list of "column aliases".

4. Using `FileProvider` to Capture a Full-Sized Photo

Once again, the lecture notes spell out how to do this. There's a lot of fiddly details in the use of `FileProvider`, but (for the purposes of the unit) you're only expected to understand the *concepts* here.

You can copy and paste the XML and Java code from the lecture notes, just making sure to select appropriate values for:

- The provider authority (`android:authorities="..."` in XML). This is supposed to be unique across all providers on the device, so the convention is to use a package-like string of the form "org.example.myapp.fileprovider".
- The request code. Again, this should be different from whatever other request codes your app might be using.
- The photo filename.

As before, display the returned photo in a `ImageView` element.

Finally, once you've tested this, let's take a look at the photo file directly (just to prove it's there). It should have been written to a file in your app's storage area, and Android Studio allows us to browse those files. Go to "View" → "Tool Windows" → "Device File Explorer".

An app's local data is stored in `/data/data/`*app-package*`.`*app-name*`/files/`. There will be a lot of existing apps to scroll through, but you should be able to find yours based on its package name. Then, see if you can find the photo file.

End of Worksheet