Lab 2 – Extending the UI with Multiple Activities and Intents

Getting started

This is the second in a series of labs that allow you to develop the MyRuns App. The goal of the app is to capture and display (using maps) walks and runs using your Android phone.

The second lab introduces extensions to the UI (e.g., new views such as tabs and ActivityPeference and category), expands the number of activities to deal with the new screens we design, and importantly, exposes us to intents and interaction with built-in apps. The new version of the MyRuns App will invoke built-in apps such as the browser, camera and galley. Our activities will use intents as a means to pass information between activities; for example we will "fire" the android built-in web browser to display our class project page; use intents to fire the camera app which will pass back an image to the calling activity.

This lab extends lab 1. So to start with create a copy of lab 1 and rename/refactor it and call it lab 2. That way you can extend the code in lab 2 and keep your lab 1 untouched.

We are going to start this lab by having you download the lab 2 app from my website and install it on your emulator. Once you install it you can run it and play with the app. You can get a good sense of what is needed since your code will have to functionally be the same – you can change the style or do more things but the downloaded app serves as a baseline for the design of your lab 2.

This is a challenging programming exercise. You will have to do some digging and self-learning using the book and web to craft your solution.

OK. Let's get started.

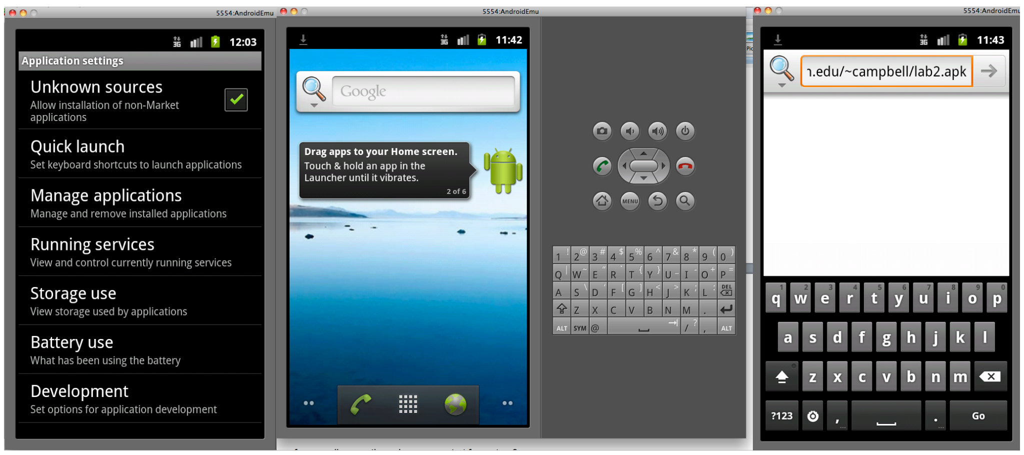
Play with the real app

You can run lab 2 by downloading the .apk file using the browser on your emulator and installing it. Here are detailed set of instructions and screen shots to work your way through the installation – once done it's a breeze the next time.

1) Run your emulator and go to the home screen (you can do this by clicking the home button on the emulator) as shown in Figure 1(b).

2) First go to Settings and select Applications Settings and select unknown sources – you do this so you can download the app over the network (from my website) to your phone (well your emulator) – in the next step.

3) Hit the home button again as Figure 1(b). You should see the Google search box. Type in the following URL into Google: www.cs.dartmouth.edu/~campbell/lab2.apk and the hit the Go key, as shown in Figure 1(c). The app should download. You should see the download icon in the status bar appear, as shown in Figure 1(c).



Application settings

Unknown sources
Allow installation of non-Market applications ✓

Quick launch
Set keyboard shortcuts to launch applications

Manage applications
Manage and remove installed applications

Running services
View and control currently running services

Storage use
View storage used by applications

Battery use
What has been using the battery

Development
Set options for application development

Drag apps to your Home screen.
Touch & hold an app in the Launcher until it vibrates.
2 of 6

ı.edu/~campbell/lab2.apk

Figure 1 (a) set unknown sources; (b) hit home button; (c) enter the URL hit Go

4) Now go to the download directory as shown in Figure 1(d) and you will see the downloaded file called lab2.apk. Click on the icon of the file to install as shown in Figure 1(e). It might prompt you for some answers, hit install. After the app is installed you will get a screen like Figure 1(f).

5) You can click done and go fine the app called MyRunsLab2 and click it or just hit open to start the app. Assume you hit done. Now hit the home button again and you will go back to home as shown in Figure 1(b). Now you want to find the app MyRunsLab2 to start it. Click on the button between the phone and browser icons – looks like a matrix of dots. That brings you to a view of all the apps on your phone. Now tab down on that screen and you will find you MyRunLab2 app.

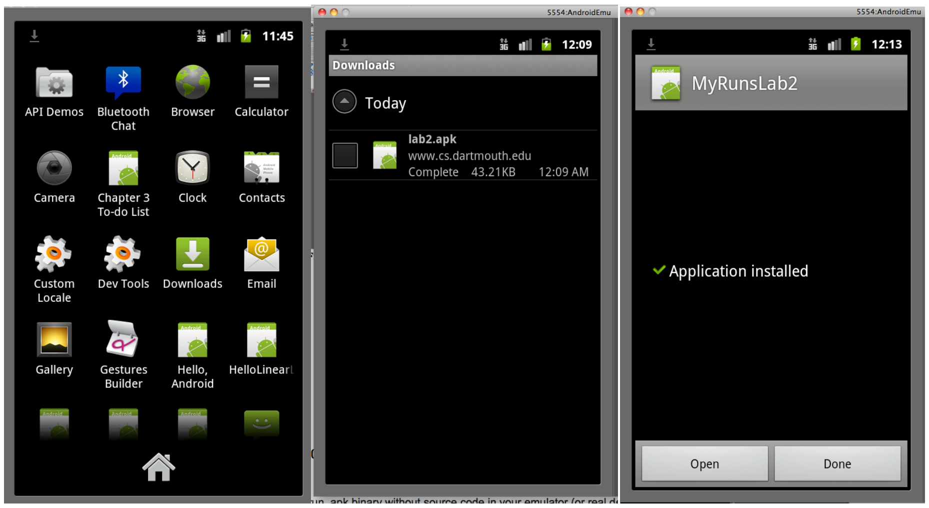


Figure 1 (d) go to downloads; (e) click on lab2.apk icon to install; (f) hit done or open

5) Click MyRunLab2 to start the app. You can now run the actual lab 2 and understand in detail what the UI is and how the components work together. This is a great way to start designing and coding up your app – sort of reserve engineering the app. We will add lots of details for helping you do the lab as well.

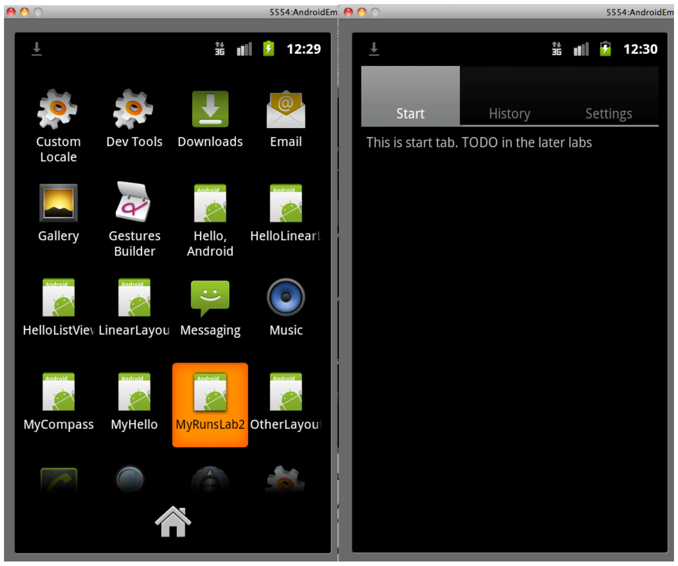


Figure 1 (g) navigate to the app; (h) start it – no play with it

Create a new lab

We will create multiple activities to handle the extended UI – the figure 2 below shows the names of all the activities we need to code. The diagram also shows that we ill be creating some new layout and xml files – we will come back to those.

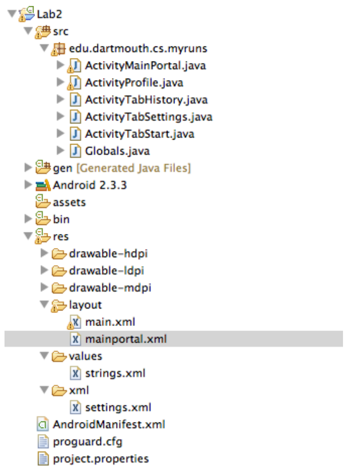
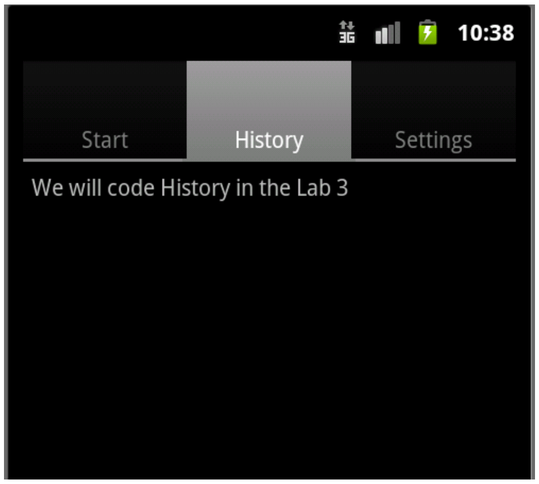

```
▼ Lab2
  ▼ src
    ▼ edu.dartmouth.cs.myruns
      ▶ ActivityMainPortal.java
      ▶ ActivityProfile.java
      ▶ ActivityTabHistory.java
      ▶ ActivityTabSettings.java
      ▶ ActivityTabStart.java
      ▶ Globals.java
  ▶ gen [Generated Java Files]
  ▶ Android 2.3.3
    assets
  ▶ bin
  ▼ res
    ▶ drawable-hdpi
    ▶ drawable-ldpi
    ▶ drawable-mdpi
    ▼ layout
        main.xml
        mainportal.xml
    ▼ values
        strings.xml
    ▼ xml
        settings.xml
    AndroidManifest.xml
    proguard.cfg
    project.properties
```

Figure 2: Lab 2 Project Files

Create the tab activities

Create an activity to manage the main portal that implements the tabs as shown in Figure 3 below.



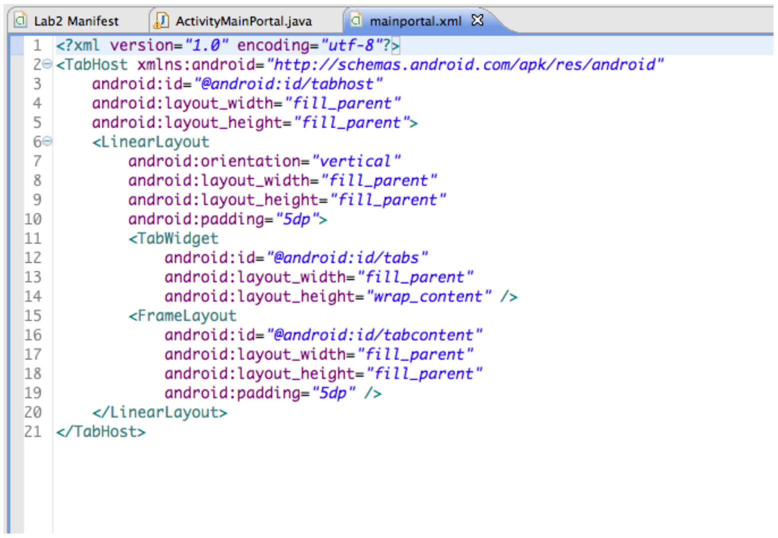
We will code History in the Lab 3

Figure 3: The ActivityMainPortal used three tabs

Setting up tabs in similar to the views tutorial (but in that case nice icons were used for the tabs – you can do that if you wish). Checkout the tutorial again:

http://developer.android.com/resources/tutorials/views/hello-tabwidget.html

This activity will render the tab layout captured in mainportal.xml, as shown below. mainportal.xml uses a TabHost and a TabWidget to create a tabbed UI. TabHost must be the root node for the layout. It contains TabWidget for displaying the tabs and a FrameLayout for displaying the tab content. Feel free to use the xml below, which is the same as in the tutorial, linked above. You will need to create the mainportal.xml in layouts.

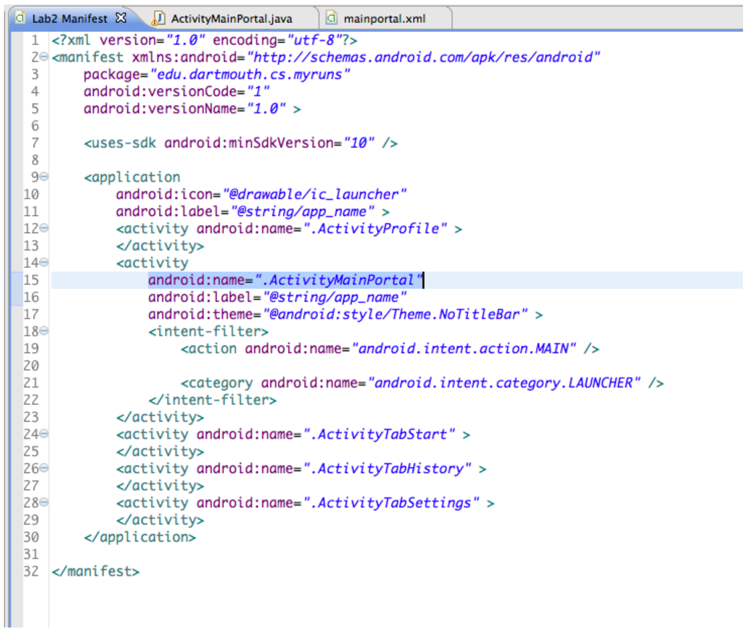


Lab2 Manifest | ActivityMainPortal.java | mainportal.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="5dp">
        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:padding="5dp" />
    </LinearLayout>
</TabHost>
```

Figure 4: mainportal.xml shows the TabHost

As mentioned above you create 3 activities one to manage each tab: ActivityTabHistory, ActivityTabSettings and ActivityTabStart. To create these activities we need to update the Manifest, as below. Note that each of these activities is defined in the Manifest.



Lab2 Manifest | ActivityMainPortal.java | mainportal.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.dartmouth.cs.myruns"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="10" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity android:name=".ActivityProfile" >
        </activity>
        <activity
            android:name=".ActivityMainPortal"
            android:label="@string/app_name"
            android:theme="@android:style/Theme.NoTitleBar" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".ActivityTabStart" >
        </activity>
        <activity android:name=".ActivityTabHistory" >
        </activity>
        <activity android:name=".ActivityTabSettings" >
        </activity>
    </application>

</manifest>
```

Figure 5: Setting up multiple activities in the Manifest

There is also an entry for the ActivityMainPortal – which is the MAIN activity that is the first activity launched when the application is run by the android system.

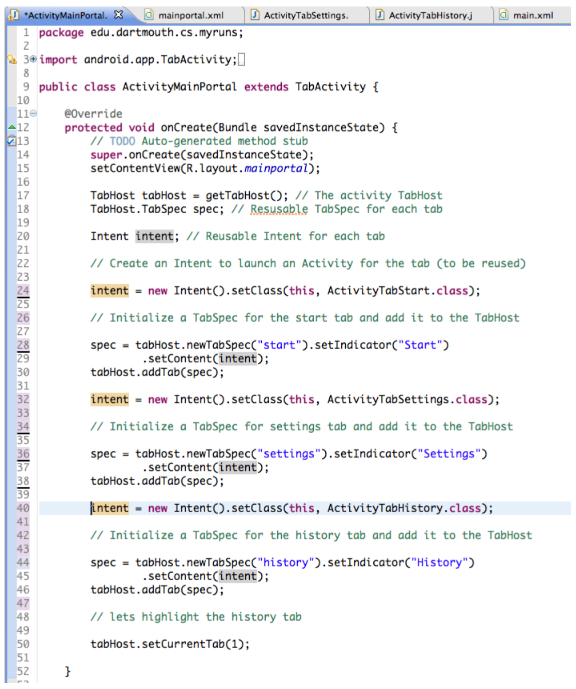
You need to create new classes for each activity. For this lab we simply create the ActivityTabHistory and ActivityTabStart activities and simply get these activities to display a message if the use clicks these tabs, as shown in the UI diagram above.

ActivityMainPortal

The ActivityMainPortal will have to render the tab views when it starts up. It will also have to create three additional activities – one for managing each view.

The heavy lifting for this lab occurs in the ActivityTabSettings activity. This activity will do a number of complex UI interactions including linking in with the ActivityProfile that we created in Lab 1 (note we extend this activity in a fairly significant manner) and creating a settings screen (which will include a number of new concepts). So create new classes for all these activities. A good idea is to use a Log.d() at the start of OnCreate for these activities.

Note, that some of these activities extend other root classes than the base Activity class used for ActivityProfile; for example, the ActivityMainPortal extends TabActivity – makes sense because it handles the tab widget layout, as shown in the Figure below.



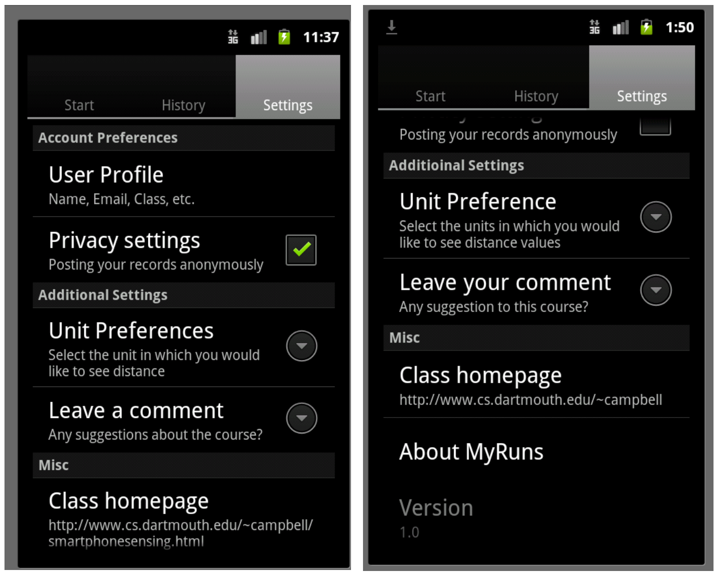
```java
package edu.dartmouth.cs.myruns;

import android.app.TabActivity;

public class ActivityMainPortal extends TabActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.mainportal);

        TabHost tabHost = getTabHost(); // The activity TabHost
        TabHost.TabSpec spec; // Resusable TabSpec for each tab

        Intent intent; // Reusable Intent for each tab

        // Create an Intent to launch an Activity for the tab (to be reused)

        intent = new Intent().setClass(this, ActivityTabStart.class);

        // Initialize a TabSpec for the start tab and add it to the TabHost

        spec = tabHost.newTabSpec("start").setIndicator("Start")
                .setContent(intent);
        tabHost.addTab(spec);

        intent = new Intent().setClass(this, ActivityTabSettings.class);

        // Initialize a TabSpec for settings tab and add it to the TabHost

        spec = tabHost.newTabSpec("settings").setIndicator("Settings")
                .setContent(intent);
        tabHost.addTab(spec);

        intent = new Intent().setClass(this, ActivityTabHistory.class);

        // Initialize a TabSpec for the history tab and add it to the TabHost

        spec = tabHost.newTabSpec("history").setIndicator("History")
                .setContent(intent);
        tabHost.addTab(spec);

        // lets highlight the history tab

        tabHost.setCurrentTab(1);

    }
```

Figure 6: The ActivityMainPortal updates the spec of the TabHost object with intents to start each activity

Things to note about the code snippet: first, it renders the view from mainportal and then creates a TabHost and a spec for each of the three tabs. An intent is created to fire each activity. This means that if the user clicks a certain tab then the intent that is set up during initialization -- in OnCreate when the ActivityMainPortal is created -- will be used to start the respective activity stored in the spec added to the TabHost object. There are three specs added to the object and each activity name is added by name so it can be correctly fired.

ActivityTabSettings

The layout design for settings.xml that uses ActivityPreference and category is shown in Figure 7 below. We need to use the xml file to code up the preferences for the settings UI shown in Figure 1. In this section, we will first discuss how to design the screen and then discuss each of the user input options in turn.



Account Preferences

User Profile
Name, Email, Class, etc.

Privacy settings
Posting your records anonymously

Additional Settings

Unit Preferences
Select the unit in which you would like to see distance

Leave a comment
Any suggestions about the course?

Misc

Class homepage
http://www.cs.dartmouth.edu/~campbell/smartphonesensing.html

Posting your records anonymously

Additioinal Settings

Unit Preference
Select the units in which you would like to see distance values

Leave your comment
Any suggestion to this course?

Misc

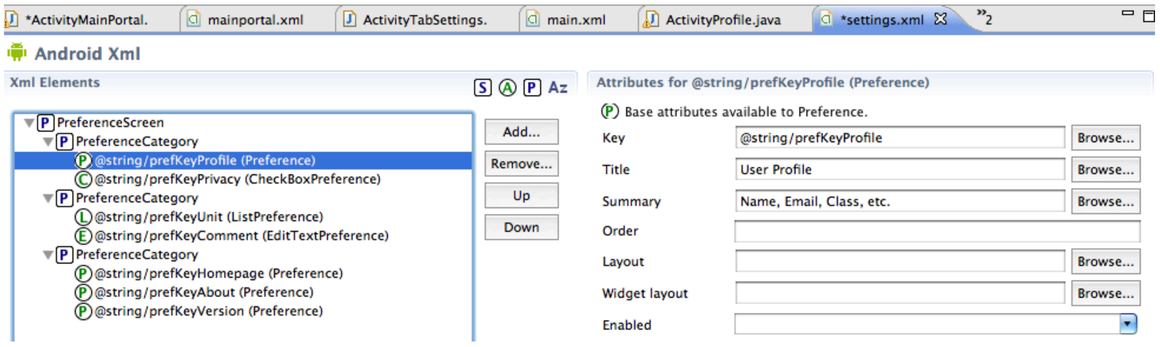
Class homepage
http://www.cs.dartmouth.edu/~campbell

About MyRuns

Version
1.0

Figure 7: Settings view using preference category

Setting up preferences: Looking at the settings.xml below you can build your screen using the structure or xml views. In structure you can build your categories by adding PreferenceCategory to the PreferenceScreen. On the right you fill in the key, title and summary. The key is used to refer to the particular category for example prefKeyProfile. You can use that key to for example fire an activity if the user selects the category



Xml Elements

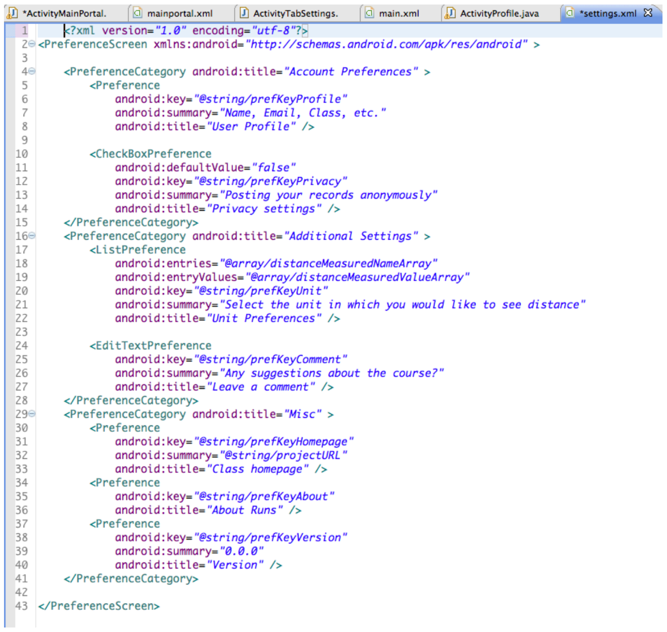
- PreferenceScreen
  - PreferenceCategory
    - @string/prefKeyProfile (Preference)
    - @string/prefKeyPrivacy (CheckBoxPreference)
  - PreferenceCategory
    - @string/prefKeyUnit (ListPreference)
    - @string/prefKeyComment (EditTextPreference)
  - PreferenceCategory
    - @string/prefKeyHomepage (Preference)
    - @string/prefKeyAbout (Preference)
    - @string/prefKeyVersion (Preference)

Attributes for @string/prefKeyProfile (Preference)

Base attributes available to Preference.

Key	@string/prefKeyProfile
Title	User Profile
Summary	Name, Email, Class, etc.
Order	
Layout	
Widget layout	
Enabled	

Figure 8: structure view of settings.xml: you can create a view by added different preferences



```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >

    <PreferenceCategory android:title="Account Preferences" >
        <Preference
            android:key="@string/prefKeyProfile"
            android:summary="Name, Email, Class, etc."
            android:title="User Profile" />

        <CheckBoxPreference
            android:defaultValue="false"
            android:key="@string/prefKeyPrivacy"
            android:summary="Posting your records anonymously"
            android:title="Privacy settings" />
    </PreferenceCategory>
    <PreferenceCategory android:title="Additional Settings" >
        <ListPreference
            android:entries="@array/distanceMeasuredNameArray"
            android:entryValues="@array/distanceMeasuredValueArray"
            android:key="@string/prefKeyUnit"
            android:summary="Select the unit in which you would like to see distance"
            android:title="Unit Preferences" />

        <EditTextPreference
            android:key="@string/prefKeyComment"
            android:summary="Any suggestions about the course?"
            android:title="Leave a comment" />
    </PreferenceCategory>
    <PreferenceCategory android:title="Misc" >
        <Preference
            android:key="@string/prefKeyHomepage"
            android:summary="@string/projectURL"
            android:title="Class homepage" />
        <Preference
            android:key="@string/prefKeyAbout"
            android:title="About Runs" />
        <Preference
            android:key="@string/prefKeyVersion"
            android:summary="0.0.0"
            android:title="Version" />
    </PreferenceCategory>

</PreferenceScreen>
```

Figure 9: the xml view of the structure view for settings.xml

If you flip from structure to xml view of settings.xml you can see the code representation – shown in Figure 8 and Figure 9.

For details on ActivityPreference and PreferenceCategory see:

http://developer.android.com/reference/android/preference/PreferenceActivity.html

In preference activity, Android allows you to incorporate normal preference for firing another activity, special preferences like CheckBoxPreference, EditTextPreference and ListPreference.

Preferences allow us to design more sophisticated UIs – they are different from layout views. We need to define preferences in the xml and then add code in activities to make the UI usable. The tutorial doesn't cover everything -- you can save typing quite a lot of java code by using a few tips – see Figure 10; for example, to use the preference format of creating resources you should select "preference" instead of "layout" in the Resource Type field in the dialog box when you select a "New Android XML File" when creating the settings.xml. In the same creation dialog box select "PreferenceScreen" as the root element to complete the creation of setting.xml. The setting.xml file opens in the structure viewpoint (you can switch between structure and setting.xml modes. In the structure view you can now add the necessary preferences for the ActivityTabSetting view.

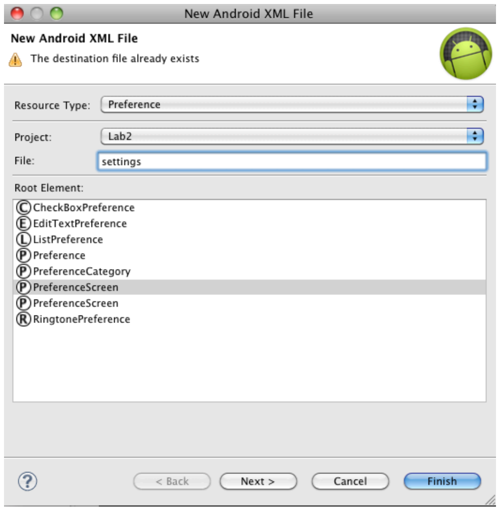


Figure 10: Creating the settings.xml file

In the "Structure" view of the generated file, you can add different type of preferences entries.

The user can select from a number of different inputs on the settings page. For example, if the user clicks on User Profile the ActivityTabSetting simply activates the ActivityProfile to handle that screen. With the exception of the Class Homepage (which invokes the android web browser) all control stays within the ActivityTabSetting activity.

Let's discuss each of the user input options.

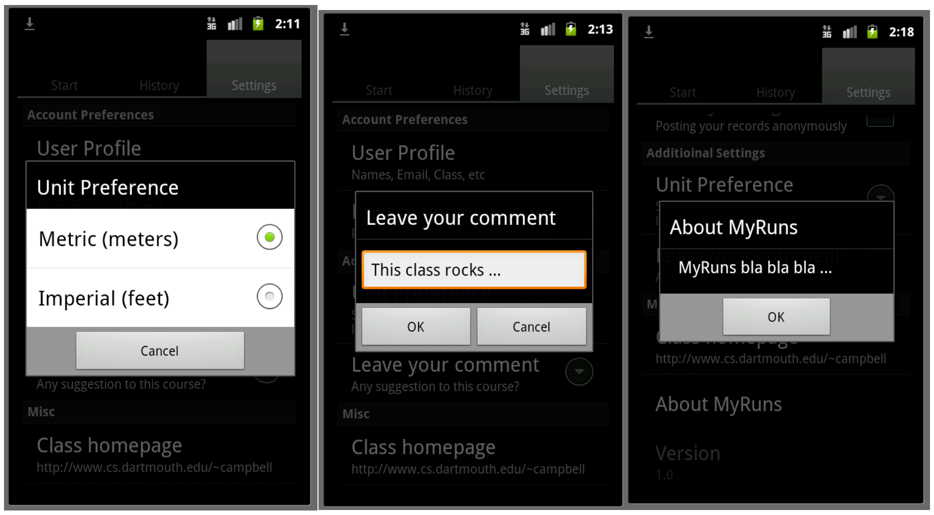


Figure 11: Unit preference; Leave a comment; About my runs

Privacy settings: This is simply a click box where the user click to set or unset the state of the box. The activity maintains this value – no need to save as context. The settings.xml has to set up CheckBoxPreference to manage this.

Unit Preference: This allows the user to enter the units of distance – km or miles. It uses a ListPreference as shown in Figure 11.

Class homepage: The ActivityTabSettings also fires the Android web browser to display the class homepage – you could use your own homepage. This is an example of our code invoking a built-in app, as shown in Figure 12.

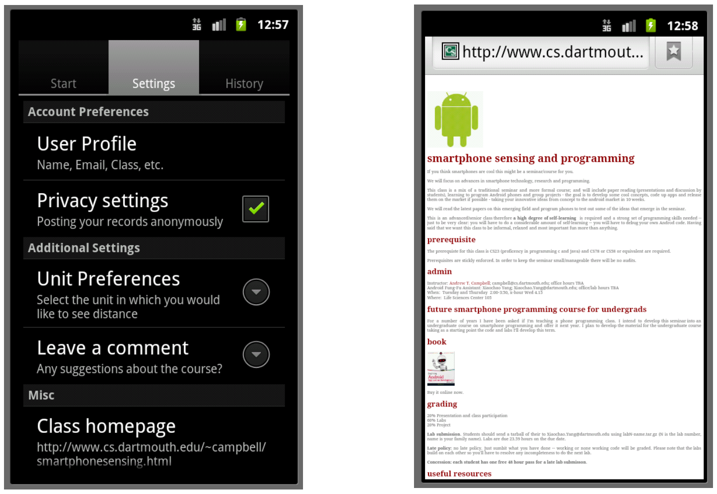
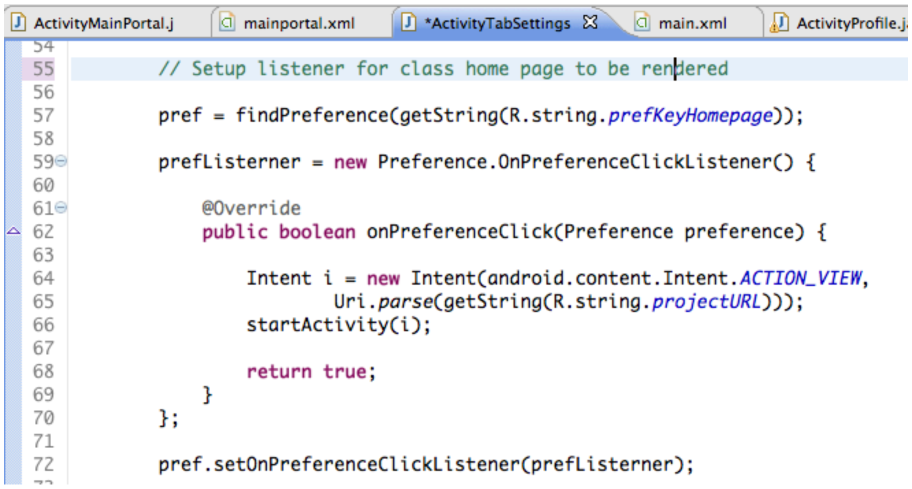

12:57
Start
Settings
History
Account Preferences
User Profile
Name, Email, Class, etc.
Privacy settings
Posting your records anonymously
Additional Settings
Unit Preferences
Select the unit in which you would like to see distance
Leave a comment
Any suggestions about the course?
Misc
Class homepage
http://www.cs.dartmouth.edu/~campbell/smartphonesensing.html
12:58
http://www.cs.dartmout...
smartphone sensing and programming
prerequisite
admin
future smartphone programming course for undergrads
book
grading
useful resources

Figure 12 Display a homepage using built-in browser

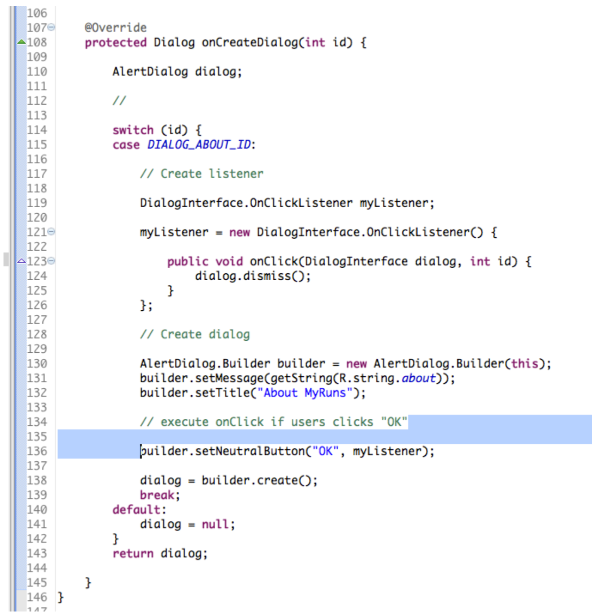
ActivityTabSettings (code snippet shown in Figure 13) uses an intent to pass the URL of the class home page to the browser to display. It creates an intent and then explicitly starts the browse using startActivity(). As discussed in class there are a number of ways to start activities depending on what you want them to do. Intents are the glue that connects activities: they allow one activity to link to another; they allow you to pass between activities; and they allow you to call other activities e.g., our activities or built in apps/activities (camera, browser, etc.)



```
        // Setup listener for class home page to be rendered

        pref = findPreference(getString(R.string.prefKeyHomepage));

        prefListerner = new Preference.OnPreferenceClickListener() {

            @Override
            public boolean onPreferenceClick(Preference preference) {

                Intent i = new Intent(android.content.Intent.ACTION_VIEW,
                        Uri.parse(getString(R.string.projectURL)));
                startActivity(i);

                return true;
            }
        };

        pref.setOnPreferenceClickListener(prefListerner);
```

Figure 13: ActivityTabSettings uses an intent and startActivity to render the class homepage

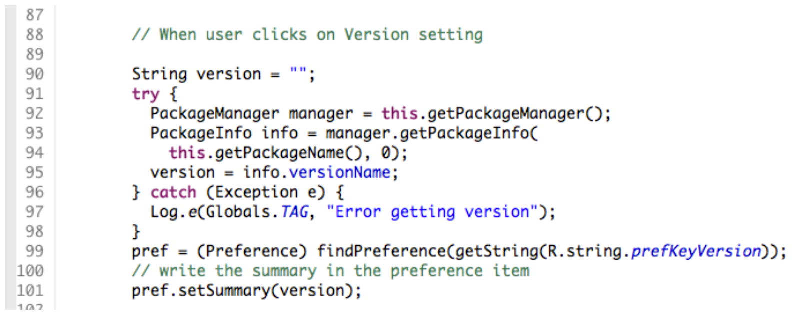
About My Runs: A dialog is created (see the discussion below on date picker dialog) and onCreateDialog customizes the dialog as shown in the code snippet below.



```
    @Override
    protected Dialog onCreateDialog(int id) {

        AlertDialog dialog;

        //

        switch (id) {
        case DIALOG_ABOUT_ID:

            // Create listener

            DialogInterface.OnClickListener myListener;

            myListener = new DialogInterface.OnClickListener() {

                public void onClick(DialogInterface dialog, int id) {
                    dialog.dismiss();
                }
            };

            // Create dialog

            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setMessage(getString(R.string.about));
            builder.setTitle("About MyRuns");

            // execute onClick if users clicks "OK"

            builder.setNeutralButton("OK", myListener);

            dialog = builder.create();
            break;
        default:
            dialog = null;
        }
        return dialog;

    }
}
```

Figure 14 Code snippet from ActivityTabSettings for About MyRuns

Version: Version is used to simply present the current version of the application code as shown in the code snippet below.



```
        // When user clicks on Version setting

        String version = "";
        try {
          PackageManager manager = this.getPackageManager();
          PackageInfo info = manager.getPackageInfo(
            this.getPackageName(), 0);
          version = info.versionName;
        } catch (Exception e) {
          Log.e(Globals.TAG, "Error getting version");
        }
        pref = (Preference) findPreference(getString(R.string.prefKeyVersion));
        // write the summary in the preference item
        pref.setSummary(version);
```

Figure 15 Code snippet from ActivityTabSettings for Version

ActivityProfile

Part of this lab requires that you extend the ActivityProfile that you created in lab 1; specifically, we need to add:

- Birthday input using a dialog and date picker for input. Your birthday needs to be then written to the screen as shown in Figure 16.
- Profile image where you can take a picture or select a photo from the galley and render it. This requires the ActivityProfile to invoke built-in apps such as the camera and the galley. In addition, the image provided by the camera and galley needs to be cropped to fit the dimensions of the image on the screen.
- The view for the ActivityProfile screen is scrollable (which means that the use can scroll up and down through the various views – the birthday to the

type of major). However, the save and cancel buttons remain static at the bottom of the screen, as shown in Figure 16.

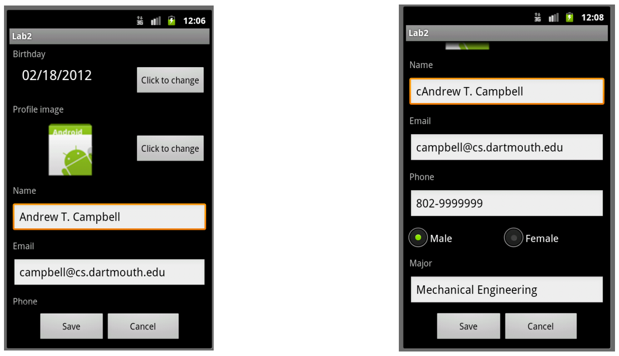
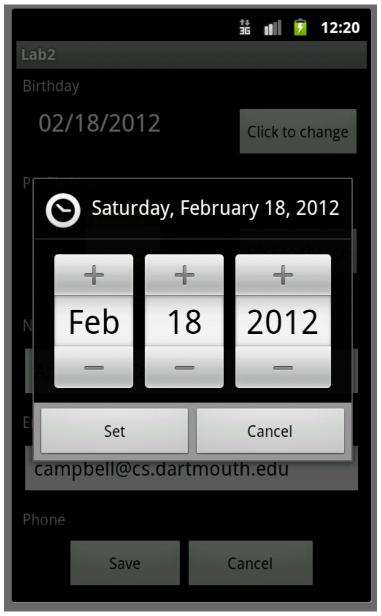
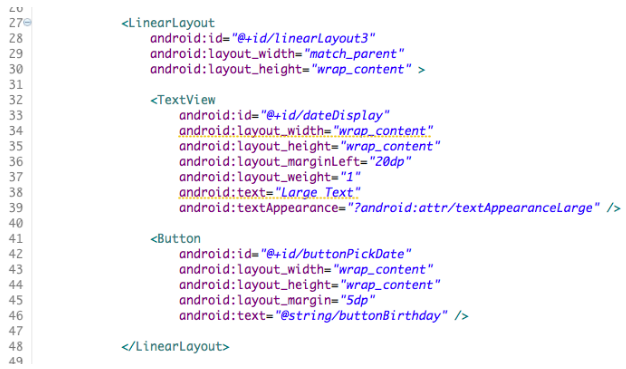


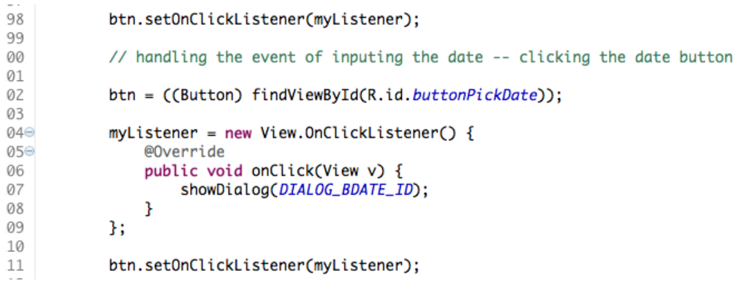
Figure 16: ActivityProfile UI defined in main.xml: The user can select and image and input their birthday

Birthday UI: Figure 17 below shows the UI design for birthday input. The snippet of main.xml show that a text view and button is defined. When the user clicks on Click to change the catches the event and displays the dialog. ShowDialog() is used to set this on in the on click listener code.





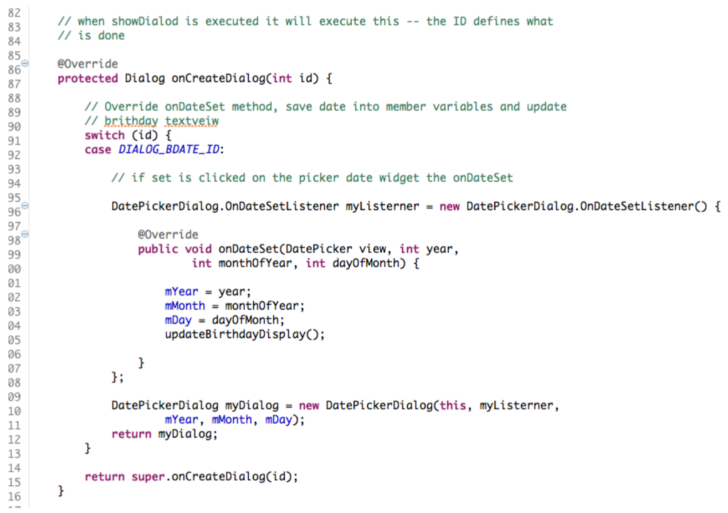
```
<LinearLayout
    android:id="@+id/linearLayout3"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

    <TextView
        android:id="@+id/dateDisplay"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="20dp"
        android:layout_weight="1"
        android:text="Large Text"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <Button
        android:id="@+id/buttonPickDate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:text="@string/buttonBirthday" />

</LinearLayout>
```



```
btn.setOnClickListener(myListener);

// handling the event of inputing the date -- clicking the date button

btn = ((Button) findViewById(R.id.buttonPickDate));

myListener = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        showDialog(DIALOG_BDATE_ID);
    }
};

btn.setOnClickListener(myListener);
```

Figure 17: Birthday UI, snippet on main.xml and listener code

When the showDialog() method executes it calls onCreateDialog(), which renders the picker and saves and updates the birthday input using updateBirthday(), which you also have to code up.



```
// when showDialod is executed it will execute this -- the ID defines what
// is done

@Override
protected Dialog onCreateDialog(int id) {

    // Override onDateSet method, save date into member variables and update
    // brithday textveiw
    switch (id) {
    case DIALOG_BDATE_ID:

        // if set is clicked on the picker date widget the onDateSet

        DatePickerDialog.OnDateSetListener myListerner = new DatePickerDialog.OnDateSetListener() {

            @Override
            public void onDateSet(DatePicker view, int year,
                    int monthOfYear, int dayOfMonth) {

                mYear = year;
                mMonth = monthOfYear;
                mDay = dayOfMonth;
                updateBirthdayDisplay();

            }
        };

        DatePickerDialog myDialog = new DatePickerDialog(this, myListerner,
                mYear, mMonth, mDay);
        return myDialog;
    }

    return super.onCreateDialog(id);
}
```

Figure 18: OnCreatDialog() for DatePicker

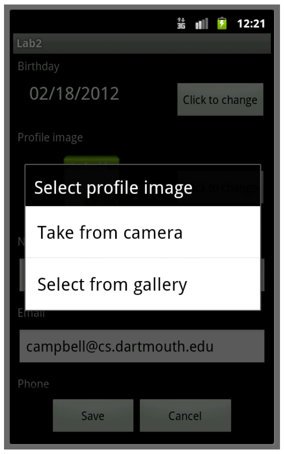
Image Profile UI: A very cool part of Android is using intents to invoke built-in apps to work with our application. Here we use the camera and the chooser (galley) to provide services: that is, the user can take a picture or select a picture from their stored photos/images and use the image for their profile.

We discussed how activities use intents to invoke other activities. In this case we want to pass information between activities and in the case of the camera and galley information is passed back from the built-in apps – that is the image. So we need to write the code to invoke built-ins and get data back from them.

Note, this part of the lab is the trickiest and will require you to apply effort to get it to work correctly.

Here is a link to a resource that is very useful:

http://www.londatiga.net/featured-articles/how-to-select-and-crop-image-on-android/



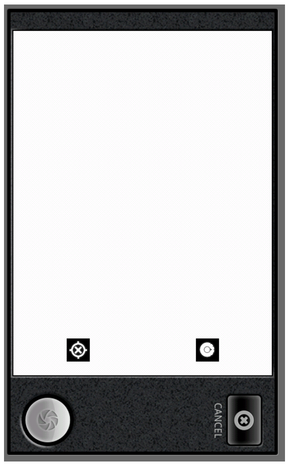
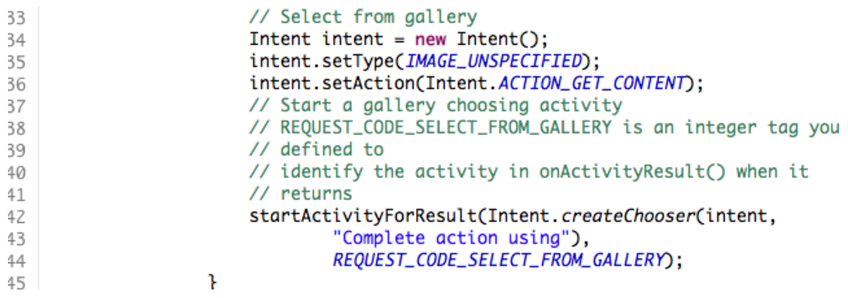


Figure 19 the dialog allows the user to select options – e.g., the camera

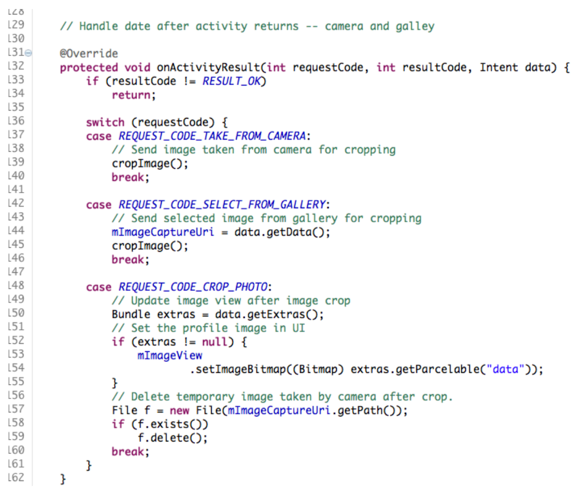
The camera and galley (and crop activity that is also use) pass data back to the calling activity – that is ActivityProfile; for example, the galley activity needs to pass back the chosen picture to the ActivityProfile activity -- here is the setup with startActivityForResults()



```
                // Select from gallery
                Intent intent = new Intent();
                intent.setType(IMAGE_UNSPECIFIED);
                intent.setAction(Intent.ACTION_GET_CONTENT);
                // Start a gallery choosing activity
                // REQUEST_CODE_SELECT_FROM_GALLERY is an integer tag you
                // defined to
                // identify the activity in onActivityResult() when it
                // returns
                startActivityForResult(Intent.createChooser(intent,
                        "Complete action using"),
                        REQUEST_CODE_SELECT_FROM_GALLERY);
            }
```

Figure 20 startActivityForResult() used for galley

You have to implement an onActivityResult method to get data back from the called activity; for example, the galley passes back a data object (the image), which we crop to size. The code snippet below from ActivityProfile is a single entry (or callback) from activities that are started by ActivityProfile – that is, camera, galley and crop. The code shows how onActivityResults() deals with the call back from these built-in apps.



```java
// Handle date after activity returns -- camera and galley

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != RESULT_OK)
        return;

    switch (requestCode) {
    case REQUEST_CODE_TAKE_FROM_CAMERA:
        // Send image taken from camera for cropping
        cropImage();
        break;

    case REQUEST_CODE_SELECT_FROM_GALLERY:
        // Send selected image from gallery for cropping
        mImageCaptureUri = data.getData();
        cropImage();
        break;

    case REQUEST_CODE_CROP_PHOTO:
        // Update image view after image crop
        Bundle extras = data.getExtras();
        // Set the profile image in UI
        if (extras != null) {
            mImageView
                    .setImageBitmap((Bitmap) extras.getParcelable("data"));
        }
        // Delete temporary image taken by camera after crop.
        File f = new File(mImageCaptureUri.getPath());
        if (f.exists())
            f.delete();
        break;
    }
}
```

Figure 21 Snippet from ActivityProfile for handling call backs from built-in apps

Tips

Don't miss this step: Download the real app to your emulator and play with it

Use command shift f to correctly format your code
Use command shift o to import classes automatically

Reading: The links in the lab are important to read plus read chapter two in the book on Intents:

- Read Intents and Intent filters on the dev site
- http://developer.android.com/guide/topics/intents/intents-filters.html

Again, to complete these assignments you will need to sort a number or problems out and self-learning is needed.

Debugging: Use Log.d() as printf style debugging to start with. It helps answer the obvious problems. You can print out data using Log.d(). More sophisticated debugging would help in addition start getting use to the messages printed out on LogCat.