Android Tutorial

Larry Walters
OOSE Fall 2011
This tutorial is a brief overview of some major concepts...Android is much richer and more complex

Developer’s Guide

API Reference
Tools

- **Phone**
  - Android Plugin (ADT)
  - Install everything except Additional SDK Platforms, unless you want to
Android SDK

- Once installed open the SDK Manager
- Install the desired packages
- Create an Android Virtual Device (AVD)
AVD
In Eclipse, go to Help -> Install New Software
Click ‘Add’ in top right
Enter:
- Name: ADT Plugin
- Location: https://dl-ssl.google.com/android/eclipse/
Click OK, then select ‘Developer Tools’, click Next
Click Next and then Finish
Afterwards, restart Eclipse
Specify SDK location (next 3 slides)
- Must do this every time start a new project in a new location (at least in Windows)
ADT Plugin (2)
ADT Plugin (3)
ADT Plugin (4)

Preferences dialog box with options for Android preferences including SDK location and list of SDK targets. The SDK location is set to C:\Program Files\Android\android-sdk. The list of SDK targets includes Android 1.5, Android 2.1-update1, Android 2.2, Android 2.3.1, Android 2.3.3, Android 3.0, Android 3.1, and Android 3.2. Each target has a corresponding vendor and platform version number.
Creating a Project (1)
Creating a Project (2)

Need the items circled

Then click Finish
Project Components

- src – your source code
- gen – auto-generated code (usually just R.java)
- Included libraries
- Resources
  - Drawables (like .png images)
  - Layouts
  - Values (like strings)
- Manifest file
**XML**

- Used to define some of the resources
  - Layouts (UI)
  - Strings
- Manifest file
- Shouldn’t usually have to edit it directly, Eclipse can do that for you
- Preferred way of creating UIs
  - Separates the description of the layout from any actual code that controls it
  - Can easily take a UI from one platform to another
**R Class**

- Auto-generated: you shouldn’t edit it
- Contains IDs of the project resources
- Enforces good software engineering
- Use findViewById and Resources object to get access to the resources
  - Ex. Button b = (Button)findViewById(R.id.button1)
  - Ex. getResources().getString(R.string.hello));
Eclipse has a great UI creator
- Generates the XML for you

Composed of View objects

Can be specified for portrait and landscape mode
- Use same file name, so can make completely different UIs for the orientations without modifying any code
Layouts (2)
Click ‘Create’ to make layout modifications

When in portrait mode can select ‘Portrait’ to make a res sub folder for portrait layouts
- Likewise for Landscape layouts while in landscape mode
- Will create folders titled ‘layout-port’ and ‘layout-land’

Note: these ‘port’ and ‘land’ folders are examples of ‘alternate layouts’, see here for more info
- [http://developer.android.com/guide/topics/resources/providing-resources.html](http://developer.android.com/guide/topics/resources/providing-resources.html)

Avoid errors by making sure components have the same id in both orientations, and that you’ve tested each orientation thoroughly
Strings

- In res/values
  - strings.xml
- Application wide available strings
- Promotes good software engineering
- UI components made in the UI editor should have text defined in strings.xml

- Strings are just one kind of ‘Value’ there are many others
Manifest File (1)

- Contains characteristics about your application
- When have more than one Activity in app, NEED to specify it in manifest file
  - Go to graphical view of the manifest file
  - Add an Activity in the bottom right
  - Browse for the name of the activity
- Need to specify Services and other components too
- Also important to define permissions and external libraries, like Google Maps API
Manifest File (2) – Adding an Activity

The activity tag declares an **android.app.Activity** class that is available as part of the package’s application components, implementing a part of the application’s user interface.
Android Programming Components

- Activity

- Service
  - [http://developer.android.com/guide/topics/fundamentals/services.html](http://developer.android.com/guide/topics/fundamentals/services.html)

- Content Providers

- Broadcast Receivers

- Android in a nutshell:
Activities (1)

- The basis of android applications
- A single Activity defines a single viewable screen
  - the actions, not the layout
- Can have multiple per application
- Each is a separate entity
- They have a structured life cycle
  - Different events in their life happen either via the user touching buttons or programmatically
Activities

1. Activity starts
2. onCreate()
3. onStart()
4. onResume()
5. onPause()
6. onStop()
7. onDestroy()

- User navigates back to the activity
- Process is killed
- Other applications need memory
- The activity is no longer visible
- The activity comes to the foreground

Activity is shut down
## Services (1)

- **Run in the background**
  - Can continue even if Activity that started it dies
  - Should be used if something needs to be done while the user is not interacting with application
    - Otherwise, a thread is probably more applicable
  - Should create a new thread in the service to do work in, since the service runs in the main thread

- **Can be bound to an application**
  - In which case will terminate when all applications bound to it unbind
  - Allows multiple applications to communicate with it via a common interface

- **Needs to be declared in manifest file**

- **Like Activities, has a structured life cycle**
Services (2)

Component calls `startService()`

- `onCreate()`
- `onStartCommand()`
- Service is running
  - The service is stopped by itself or a client
  - `onDestroy()`
  - Service is shut down

Unbounded

Component calls `bindService()`

- `onCreate()`
- `onBind()`
- Service is running (clients are bound to it)
  - All clients unbind by calling `unbindService()`
  - `onUnbind()`
  - `onDestroy()`
  - Service is shut down

Bounded
Running in Eclipse (1)

- Similar to launching a regular Java app, use the launch configurations
- Specify an Android Application and create a new one
- Specify activity to be run
- Can select a manual option, so each time program is run, you are asked whether you want to use the actual phone or the emulator
  - Otherwise, it should be smart and use whichever one is available
Running in Eclipse (2)
Running in Eclipse (3)
Running in Eclipse (4)
USB Debugging

- Should be enabled on phone to use developer features
- In the main apps screen select Settings -> Applications -> Development -> USB debugging (it needs to be checked)
Android Debug Bridge

- Used for a wide variety of developer tasks
  - Read from the log file
  - Show what android devices are available
  - Install android applications (.apk files)
- In the ‘platform-tools’ directory of the main android sdk directory
  - Recommend putting this directory and the ‘tools’ directory on the system path
- adb.exe
Debugging

- Instead of using traditional System.out.println, use the Log class
  - Imported with android.util.Log
  - Multiple types of output (debug, warning, error, …)
  - Log.d(<tag>,<string>)

- Can be read using logcat.
  - Print out the whole log, which auto-updates
    - adb logcat
  - Erase log
    - adb logcat –c
  - Filter output via tags
    - adb logcat <tag>:<msg type> *:S
    - can have multiple <tag>:<msg type> filters
    - <msg type> corresponds to debug, warning, error, etc.
    - If use Log.d(), then <msg type> = D

- Reference
Some say you need to root the phone – that is not true

One option: Android Screen Capture

- It’s slow, but fine for screenshots of applications whose screens aren’t changing fast
- Read their installation help, following the extra steps if need be (I had to copy adb.exe and some dll files, as they explain)
Maps Example (1)

- Using Google Maps in your app
- Setup project to use ‘Google API’ version
- Edit Manifest file
  - To indicate the app will use maps and the internet
- Get a maps API key
- Note: Google Maps API can display a map and draw overlays, but is not the full Google Maps experience you enjoy on the web
  - For example, there does not seem to be inherent support for drawing routes between points (if you find it let me know)...however, you can draw lines between points and almost any type of overlay, but that’s different than street routes
  - The directions API is a web service, which is different, among several other Google web services
- Read the Google API terms of use
Maps Example (2)
Maps Example (3) – Manifest (1)

- Open Manifest file
- Add map library tag
  - Add the ‘Uses Library’ com.google.android.maps
- Indicate the app will access the internet
  - Add the ‘Permission’ android.permission.INTERNET
- End goal is to add the following two lines to XML file, under the <manifest> and <application tags>, respectively
  - Under the <manifest> tag
    - <uses-permission android:name="android.permission.INTERNET"></uses-permission>
  - Under the <application> tag
    - <uses-library android:name="com.google.android.maps"></uses-library>
- Following is GUI way to add them
Maps Example (4) – Manifest (2)
Maps Example (5) – Manifest (3)

- Select ‘Add’ under ‘Uses Library’ (last slide)
- Then select ‘Uses Library at this prompt
- Set name as: com.google.android.maps (next slide) and save
Maps Example (6) – Manifest (4)
Maps Example (7) – Manifest (5)
Select ‘Permissions’ and then ‘Add’ (last slide)
Select ‘Uses Permissions’ at this prompt
Set name to: android.permission.INTERNET and save (next slide)
Maps Example (9) – Manifest (7)

The *uses-permission* tag requests a "permission" that the containing package must be granted in order for it to operate correctly.

Name: android.permission.INTERNET
Maps Example (10) – Maps API Key (1)

- All Android applications need to be signed
  - The debug mode signs for you with special debug certificate
- All MapView elements in map applications need to have an API key associated with them
  - That key must be registered with the certificate used to sign the app
- When releasing app, need to sign with a release certificate and get a new API Key
For debug mode, get the MD5 fingerprint of the debug certificate
  - Locate the ‘keystore’
    - Windows Vista: C:\Users\<user>\.android\debug.keystore
    - Windows XP: C:\Documents and Settings\<user>\.android\debug.keystore
    - OS X and Linux: ~/.android/debug.keystore
  - Use Keytool (comes with Java, in the bin directory with the other Java tools, should put that dir on system PATH) to get fingerprint
    - keytool -list –v -alias androiddebugkey -keystore “<path_to_debug_keystore>” -storepass android -keypass android
      - If don’t include –v option, then will probably get only 1 fingerprint, and if it’s not MD5, then need –v (Java 7 needs –v)
      - Extract the MD5 fingerprint, SHA will not work unfortunately
  - Go to https://code.google.com/android/maps-api-signup.html, agree to terms and paste MD5 fingerprint, you will then be given an API Key
Maps Example (12)

- Need to put MapView tag in XML
  - com.google.android.maps.MapView
  - MapView is the basic view that represents a Google Map display
  - Must include API Key in XML, inside a layout
    - <com.google.android.maps.MapView
      android:id="@+id/mapview"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:clickable="true"
      android:apiKey="<api key>"/>

- Maps API Reference
Acknowledgements

- Android Developer’s Website
  - Activity and Service life-cycle flow charts
  - Tons of other Android info
- Google Maps API external library
- MightyPocket
- Numerous Forums & other developer sites, including:
  - http://www.mail-archive.com/android-developers@googlegroups.com/msg28487.html
  - http://android.bigresource.com/ threads
  - http://groups.google.com/group/android-developers threads
  - Many http://stackoverflow.com threads
  - http://www.anddev.org/google_driving_directions_-_mapview_overlayed-t826.html
- Zainan Victor Zhou – for advice and his own tutorial