Mobile Application Development

Produced by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Android Anatomy

Anatomy of an Android App

Agenda

- ❑ In a Nutshell…
- ❑ Platform Architecture
- ❑ Application Components
- ❑ The Manifest File
- ❑ Application Resources
- ❑ The Android Application Life Cycle

Agenda

- ❑ In a Nutshell…
- ❑ Platform Architecture
- ❑ Application Components
- ❑ The Manifest File
- ❑ Application Resources
- ❑ The Android Application Life Cycle

Android Anatomy

Application Components

Application Components

- App components are the essential **building blocks** of an Android app. Each component is an entry point through which the system or a user can enter your app and some components depend on the others.
- There are four different types of app components:
  - Activities
  - Services
  - Broadcast receivers
  - Content providers
- Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

Android App Components - Activities

- An Activity is the entry point for interacting with the user. It represents a single screen with a user interface.
  - For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails.
- Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others. As such, a different app can start any one of these activities if the email app allows it.
  - For example, a camera app can start the activity in the email app that composes new mail to allow the user to share a picture

Android App Components - Activities

- An Activity facilitates the following key interactions between system and app:
  - Keeping track of what the user currently cares about (what's on screen) to ensure that the system keeps running the process that is hosting the activity.
  - Knowing that previously used processes contain things the user may return to (stopped activities), and thus more highly prioritise keeping those processes around.
  - Helping the app handle having its process killed so the user can return to activities with their previous state restored.
  - Providing a way for apps to implement user flows between each other, and for the system to coordinate these flows. (e.g. 'share')
- You implement an activity as a subclass of the **`Activity`** class. (We'll cover **`Fragment`**s in the 'Application Life Cycle' section)

Android App Components - Services

- A Service is a general-purpose entry point for keeping an app running in the background for all kinds of reasons.
- It is a component that runs in the background to perform long-running operations or to perform work for remote processes.
- A service does not provide a user interface.
  - For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity.
- Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it

Android App Components - Services

- **Started services** tell the system to keep them running until their work is completed. This could be to sync some data in the background or play music even after the user leaves the app.
- Syncing data in the background or playing music also represent two different types of **started services** that modify how the system handles them
  - Music playback is something the user is directly aware of, so the app tells the system this by saying it wants to be foreground with a notification to tell the user about it; here, the system knows that it should try really hard to keep that service's process running, because the user will be unhappy if it goes away.
  - A regular background service is not something the user is directly aware of as running, so the system has more freedom in managing its process. It may allow it to be killed (and restarted later) if it needs RAM for immediate concerns.

Android App Components - Services

- **Bound service**s run because some other app (or the system) has said that it wants to make use of the service. This is basically the service providing an API to another process.
- The system thus knows there is a dependency between these processes, so if process A is bound to a service in process B, it knows that it needs to keep process B (and its service) running for A.
- Because of their flexibility (for better or worse), services have turned out to be a really useful building block for all kinds of higher-level system concepts. Live wallpapers, notification listeners, screen savers, input methods, accessibility services, and many other core system features are all built as services that applications implement and the system binds to when they should be running.

- A service is implemented as a subclass of **`Service`**

Android App Components - Broadcast Receivers

- A **Broadcast Receiver** is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements.

- Because broadcast receivers are another well-defined entry into the app, the system can deliver broadcasts even to apps that aren't currently running.
  - So, for example, an app can schedule an alarm to post a notification to tell the user about an upcoming event... and by delivering that alarm to a Broadcast Receiver of the app, there is no need for the app to remain running until the alarm goes off.

- Many broadcasts originate from the system— e.g. a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured.

Android App Components - Broadcast Receivers

- ❑ Apps can also initiate broadcasts—for example, to let other apps know that some data has been downloaded to the device and is available for them to use.

- ❑ Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs.

- ❑ More commonly, though, a broadcast receiver is just a gateway to other components and is intended to do a very minimal amount of work. For instance, it might schedule a **`JobService`** to perform some work based on the event with **`JobScheduler`**

- ❑ A broadcast receiver is implemented as a subclass of **`BroadcastReceiver`** and each broadcast is delivered as an **`Intent`** object.

Android App Components - Content Providers

- A **Content Provider** manages a shared set of app data that you can store in the file system, in a SQLite database, on the web, or on any other persistent storage location that your app can access.

- Through the content provider, other apps can query or modify the data if the content provider allows it.

- For example, the Android system provides a content provider that manages the user's contact information.

- As such, any app with the proper permissions can query the content provider, such as **`ContactsContract.Data`**, to read and write information about a particular person

Android App Components - Content Providers

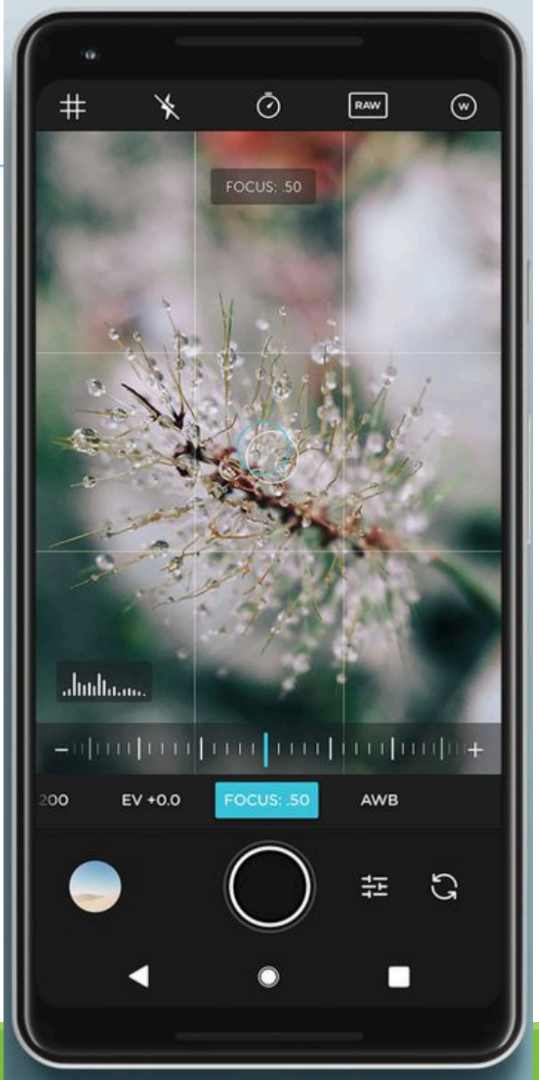
- It is tempting to think of a content provider as an abstraction on a database, because there is a lot of API and support built in to them for that common case.
- However, they have a different core purpose from a system-design perspective.
- To the system, a content provider is an entry point into an app for publishing named data items, identified by a Uniform Resource Indicator (URI) scheme.
- Thus an app can decide how it wants to map the data it contains to a URI namespace, handing out those URIs to other entities which can in turn use them to access the data.
- There are a few particular things this allows the system to do in managing an app: (next)

Android App Components - Content Providers

- Assigning a URI doesn't require that the app remain running, so URIs can persist after their owning apps have exited. The system only needs to make sure that an owning app is still running when it has to retrieve the app's data from the corresponding URI.
- These URIs also provide an important fine-grained security model. e.g, an app can place the URI for an image it has on the clipboard, but leave its content provider locked up so that other apps cannot freely access it. When a second app attempts to access that URI on the clipboard, the system can allow that app to access the data via a temporary URI permission grant so that it is allowed to access the data only behind that URI, but nothing else in the second app.
- Content providers are also useful for reading and writing data that is private to your app and not shared. E.g., the Note Pad sample app.
- A content provider is implemented as a subclass of **`ContentProvider`** and must implement a standard set of APIs that enable other apps to perform transactions.

Starting Other App Components

- ❑An aspect of the Android system design is that any app can start another app’s component.
- ❑For example, if you want the user to capture a photo with the device camera, there's probably another app that does that and your app can use it instead of developing an activity to capture a photo yourself.
- ❑You don't need to incorporate or even link to the code from the camera app. Instead, you can simply start the activity in the camera app that captures a photo. When complete, the photo is even returned to your app so you can use it.
- ❑To the user, it seems as if the camera is actually a part of your app.



App Components - Activating Components

- ❑Three of the four component types—activities, services, and broadcast receivers—are activated by an asynchronous message called an intent.
- ❑Intents bind individual components to each other at runtime. You can think of them as the messengers that request an action from other components, whether the component belongs to your app or another
- ❑An intent is created with an Intent object, which defines a message to activate either a specific component (explicit intent) or a specific type of component (implicit intent).

App Components - Activating Components

- ❑For activities and services, an intent defines the action to perform (for example, to view or send something) and may specify the URI of the data to act on, among other things that the component being started might need to know. For example, an intent might convey a request for an activity to show an image or to open a web page.

- ❑In some cases, you can start an activity to receive a result, in which case the activity also returns the result in an Intent. For example, you can issue an intent to let the user pick a personal contact and have it returned to you. The return intent includes a URI pointing to the chosen contact.

App Components - Activating Components

- ❑For broadcast receivers, the intent simply defines the announcement being broadcast. For example, a broadcast to indicate the device battery is low includes only a known action string that indicates battery is low.
- ❑Unlike activities, services, and broadcast receivers, content providers are not activated by intents. Rather, they are activated when targeted by a request from a Content Resolver.
- ❑The content resolver handles all direct transactions with the content provider so that the component that's performing transactions with the provider doesn't need to and instead calls methods on the Content Resolver object.
- ❑This leaves a layer of abstraction between the content provider and the component requesting information (for security).

App Components - Activating Components

❑ There are separate methods for activating each type of component:

- You can start an activity or give it something new to do by passing an Intent to `startActivity()` or `startActivityForResult()` (when you want the activity to return a result).
- With Android 5.0 (API level 21) and later, you can use the `JobScheduler` class to schedule actions. For earlier Android versions, you can start a service (or give new instructions to an ongoing service) by passing an Intent to `startService()`. You can bind to the service by passing an Intent to `bindService()`.
- You can initiate a broadcast by passing an Intent to methods such as `sendBroadcast()`, `sendOrderedBroadcast()`, or `sendStickyBroadcast()` .
- You can perform a query to a content provider by calling `query()` on a `ContentResolver`.

How it all Fits Together *

Big N.B. for all these!

- Based on the *Model View Controller* design pattern.
- Don't think of your program as a linear execution model:
  - Think of your program as existing in logical blocks, each of which performs some actions.
- The blocks communicate back and forth via message passing (*Intents*)
  - Added advantage, **physical user interaction (screen clicks) and inter process interaction can have the same programming interface**
  - Also the OS can bring different pieces of the app to life depending on memory needs and program use

- For each distinct logical piece of program behavior you'll write a class (derived from a base class).
- Activities/Fragments: Things the user can ***see*** on the screen. Basically, the 'controller' for each different screen in your program.
- Services: Code that isn't associated with a screen (background stuff, fairly common)
- Content providers: Provides an interface to exchange data between programs (usually SQL based)
- You'll also design your layouts (screens), with various types of widgets (Views), which is what the user sees via *Activities & Fragments*

Android Anatomy

Sources:

- https://developer.android.com/guide/platform/
- https://developer.android.com/guide/components/fundamentals
- https://www.techotopia.com/index.php/The_Anatomy_of_an_Android_Application

Thanks.