

# Android: app fundamentals

<http://developer.android.com/guide/components/fundamentals.html>

Ferruccio Damiani

Università di Torino  
[www.di.unito.it/~damiani](http://www.di.unito.it/~damiani)

Mobile Device Programming  
(Laurea Magistrale in Informatica, a.a. 2018-2019)

# Outline

- 1 APK and Manifest
- 2 High-level Concepts
- 3 The Manifest File
- 4 App Resources

# Outline

- 1 APK and Manifest
- 2 High-level Concepts
- 3 The Manifest File
- 4 App Resources

# Android application package (APK)

*Android application package (APK)* is the package file format used by the Android operating system for distribution and installation of application software. It is an archive file with an `.apk` suffix.

- The Android SDK tools compile your code—along with any data and resource files—into an APK.
- One APK file contains all the contents of an Android app.

An APK file is an archive that roughly contains:

1. An `AndroidManifest.xml` file (with precisely that name) in its root directory.
2. `Dalvik executable`.
3. `Resources`: everything that is not code (images, audio/video clips, XML files describing layouts, language packs, and so on).
4. `Native libraries`: e.g. C/C++ libraries. This is compiled code that is specific to a software layer of a processor, stored in the directory `lib` which is split into more directories, including: `armeabi` (compiled code for all ARM based processors only), `armeabi-v7a` (compiled code for all ARMv7 and above based processors only), etc.

# Android Manifest

The manifest file presents essential information about the app, information that Android system must have before it can run any of the app's code. Among other things:

- Specifies the Java package name for the app (the unique identifier for the app).
- It describes the components of the application (activities, services, broadcast receivers, and content providers). It names the classes that implement each of the components and publishes their capabilities (for example, which `Intent` messages they can handle). These declarations let the Android system know what the components are and under what conditions they can be launched.
- It determines which processes will host application components.
- It declares which permissions the application must have in order to access protected parts of the API and interact with other applications.
- It declares the permissions that others are required to have in order to interact with the application's components.
- It lists the `Instrumentation` classes that provide profiling and other information as the application is running. These declarations are present in the manifest only while the application is being developed and tested; they're removed before the application is published.
- It declares the minimum level of the Android API that the application requires.
- It lists the libraries that the application must be linked against.

# Security model

Once installed on a device, each Android app lives in its own security sandbox:

- The Android operating system is a multi-user Linux system in which each app is a different user.
- By default, the system assigns each app a unique Linux user ID (the ID is used only by the system and is unknown to the app). The system sets permissions for all the files in an app so that only the user ID assigned to that app can access them.
- Each process has its own virtual machine (VM), so an app's code runs in isolation from other apps.
- By default, every app runs in its own Linux process. Android starts the process when any of the app's components need to be executed, then shuts down the process when it's no longer needed or when the system must recover memory for other apps.

This implements the *principle of least privilege*. That is, each app, by default, has access only to the components that it requires to do its work<sup>a</sup> and no more. This aims to create a secure environment in which an app cannot access parts of the system for which it is not given permission.

---

<sup>a</sup>**Attention:** who decides what its work is?

## Cooperation and sharing

There are ways for an app to share data with other apps and for an app to access system services:

- It is possible to arrange for two apps to share the same Linux user ID, in which case they are able to access each other's files. To conserve system resources, apps with the same user ID can also arrange to run in the same Linux process and share the same VM (the apps must also be signed with the same certificate).
- An app can request permission to access device data such as the user's contacts, SMS messages, the mountable storage (SD card), camera, Bluetooth, and more. All app permissions must be granted by the user at install time.

# Outline

- 1 APK and Manifest
- 2 High-level Concepts**
- 3 The Manifest File
- 4 App Resources



# Key terminology

**Activity** Component for composing UI

**Service** Component that runs on the background

**Intent** Message element that sends actions and data to components

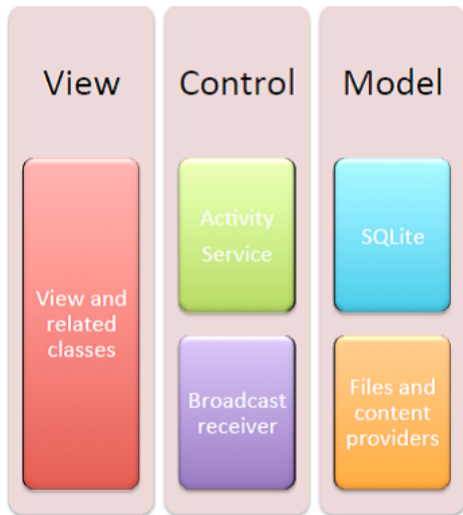
**Intent filter** Defines components by setting receiving intents

**Broadcast receiver** Receives/responds to a certain broadcast (e.g., low battery)

**Content provider** Provides standardized interfaces for sharing data among applications

**Notification** Notifies events to users

# Android through MVC



## High-level behavior (applications and activities)

- Any application can start another application's component
- When the system starts a component, it starts the process for that application and instantiates the classes needed for the component
  - ▶ If one starts the activity in the camera application that captures a photo, that activity runs in the process that belongs to the camera application, not in your application's process
- An application cannot directly activate a component from another application
  - ▶ To activate a component in another application, you must deliver a message to the system that specifies your intent to start a particular component
  - ▶ The system then activates the component for you

# Outline

- 1 APK and Manifest
- 2 High-level Concepts
- 3 The Manifest File**
- 4 App Resources

Before the Android system can start an app component, the system must know that the component exists by reading the app's `AndroidManifest.xml` file (the “manifest” file). Your app must declare all its components in this file, which must be at the root of the app project directory.

The manifest does a number of things in addition to declaring the app's components, such as:

- Identify any user permissions the app requires, such as Internet access or read-access to the user's contacts.
- Declare the minimum API Level<sup>1</sup> required by the app, based on which APIs the app uses.
- Declare hardware and software features used or required by the app, such as a camera, bluetooth services, or a multitouch screen.
- API libraries the app needs to be linked against (other than the Android framework APIs), such as the Google Maps library<sup>2</sup>.
- And more

---

<sup>1</sup>[\[http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels\]](http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels)

<sup>2</sup>[\[http://code.google.com/android/add-ons/google-apis/maps-overview.html\]](http://code.google.com/android/add-ons/google-apis/maps-overview.html)

# Declaring components

The primary task of the manifest is to inform the system about the app's components.

## Example

A manifest file can declare an activity as below.

- In the `<application>` element, the `android:icon` attribute points to resources for an icon that identifies the app.
- In the `<activity>` element, the `android:name` attribute specifies the fully qualified class name of the `Activity`<sup>a</sup> subclass and the `android:label` attributes specifies a string to use as the user-visible label for the activity.

---

<sup>a</sup>[<http://developer.android.com/reference/android/app/Activity.html>]


```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest ... >
3   <application android:icon="@drawable/app_icon.png" ... >
4     <activity android:name="com.example.project.ExampleActivity"
5               android:label="@string/example_label" ... >
6     </activity>
7     ...
8   </application>
9 </manifest>
```

You must declare all app components this way:

- `<activity>` [<http://developer.android.com/guide/topics/manifest/activity-element.html>] elements for activities
- `<service>` [<http://developer.android.com/guide/topics/manifest/service-element.html>] elements for services
- `<receiver>` [<http://developer.android.com/guide/topics/manifest/receiver-element.html>] elements for broadcast receivers
- `<provider>` [<http://developer.android.com/guide/topics/manifest/provider-element.html>] elements for content providers

Activities, services, and content providers that you include in your source but do not declare in the manifest are not visible to the system and, consequently, can never run.<sup>3</sup>

---

<sup>3</sup>However, broadcast receivers can be either declared in the manifest or created dynamically in code (as `BroadcastReceiver` objects) and registered with the system by calling `registerReceiver()`. 

## Declaring components capabilities

The real power of intents lies in the concept of *implicit intents*. An implicit intent:

- simply describes the type of action to perform (and, optionally, the data upon which you'd like to perform the action), and
- allows the system to find a component on the device that can perform the action and start it (if there are multiple components that can perform the action described by the intent, then the user selects which one to use).

The system identifies the components that can respond to an intent by comparing the intent received to the intent filters provided in the manifest file of other apps on the device.

- When you declare an activity in your app's manifest, you can optionally include intent filters that declare the capabilities of the activity so it can respond to intents from other apps.
- You can declare an intent filter for your component by adding an `<intent-filter>`<sup>4</sup> element as a child of the component's declaration element.

---

<sup>4</sup><http://developer.android.com/guide/topics/manifest/intent-filter-element.html>



## Example

if you've built an email app with an activity for composing a new email, you can declare an intent filter to respond to "send" intents (in order to send a new email) like below.

- Then, if another app creates an intent with the `ACTION_SEND`<sup>a</sup> action and pass it to `startActivity()`<sup>b</sup>, the system may start your activity so the user can draft and send an email.

<sup>a</sup>[http://developer.android.com/reference/android/content/Intent.html#ACTION\\_SEND](http://developer.android.com/reference/android/content/Intent.html#ACTION_SEND)

<sup>b</sup>[http://developer.android.com/reference/android/app/Activity.html#startActivity\(android.content.Intent\)](http://developer.android.com/reference/android/app/Activity.html#startActivity(android.content.Intent))

```
1 <manifest ... >
2   ...
3   <application ... >
4     <activity android:name="com.example.project.ComposeEmailActivity">
5       <intent-filter>
6         <action android:name="android.intent.action.SEND" />
7         <data android:type="*/*" />
8         <category android:name="android.intent.category.DEFAULT" />
9       </intent-filter>
10    </activity>
11  </application>
12 </manifest>
```

# Declaring apps requirements

There are a variety of devices powered by Android and not all of them provide the same features and capabilities.

- In order to prevent your app from being installed on devices that lack features needed by your app, it's important that you clearly define a profile for the types of devices your app supports by declaring device and software requirements in your manifest file.
- Most of these declarations are informational only and the system does not read them, but external services such as Google Play do read them in order to provide filtering for users when they search for apps from their device.

## Example

If your app requires a camera and uses APIs introduced in Android 2.1 (API Level 7), you should declare these as requirements in your manifest file like below.

- Now, devices that do not have a camera and have an Android version lower than 2.1 cannot install your app from Google Play.
- However, you can also declare that your app uses the camera, but does not require it. In that case, your app must set the `required`<sup>a</sup> attribute to “false” and check at runtime whether the device has a camera and disable any camera features as appropriate.

---

<sup>a</sup><http://developer.android.com/guide/topics/manifest/uses-feature-element.html#required>

```
1 <manifest ... >
2   <uses-feature android:name="android.hardware.camera.any"
3     android:required="true" />
4   <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="19" />
5   ...
6 </manifest>
```

# Outline

- 1 APK and Manifest
- 2 High-level Concepts
- 3 The Manifest File
- 4 App Resources

An Android app is composed of more than just code—it requires resources that are separate from the source code, such as images, audio files, and anything relating to the visual presentation of the app.

### Example

You should define animations, menus, styles, colors, and the layout of activity user interfaces with XML files.

This makes it easy to update various characteristics of your app without modifying code and—by providing sets of alternative resources—enables you to optimize your app for a variety of device configurations (such as different languages and screen sizes).

For every resource that you include in your Android project, the SDK build tools define a unique integer ID, which you can use to reference the resource from your app code or from other resources defined in XML.

## Example

If your app contains an image file named `logo.png` (saved in the `res/drawable/` directory), the SDK tools generate a resource ID named `R.drawable.logo`, which you can use to reference the image and insert it in your user interface.

One of the most important aspects of providing resources separate from your source code is the ability for you to provide alternative resources for different device configurations.

## Example

By defining UI strings in XML, you can translate the strings into other languages and save those strings in separate files. Then, based on a language qualifier that you append to the resource directory's name (such as `res/values-fr/` for French string values) and the user's language setting, the Android system applies the appropriate language strings to your UI.

Android supports many different qualifiers for your alternative resources.

The qualifier is a short string that you include in the name of your resource directories in order to define the device configuration for which those resources should be used.

## Example

You should often create different layouts for your activities, depending on the device's screen orientation and size. For example, when the device screen is in portrait orientation (tall), you might want a layout with buttons to be vertical, but when the screen is in landscape orientation (wide), the buttons should be aligned horizontally. To change the layout depending on the orientation, you can define two different layouts and apply the appropriate qualifier to each layout's directory name. Then, the system automatically applies the appropriate layout depending on the current device orientation.