IT332: Mobile Application Development

Lecture # 03: A Step Forward

Muhammad Imran

Outline

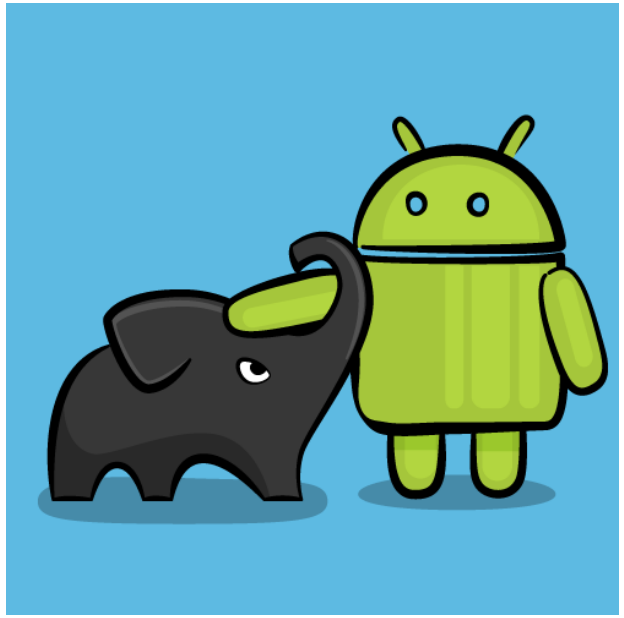
- What is Gradle
- The Build Instructions
- More on SDK Levels and Support
- Things In Common Between the Manifest and Gradle
- Working in the Manifest File
- Resources
- The Anatomy of an Android Application

What is Gradle

- Gradle is software for building software, otherwise known as “build automation software” or “build systems”.
- Other build systems in other environments, exists like make(C/C++), rake(Ruby), Ant(Java), Maven(Java), etc.
- These tools know — via intrinsic capabilities and rules that you teach them — how to determine what needs to be created (e.g., based on file changes) and how to create them.
- A build system does not compile, link, package, etc. applications directly, but instead directs separate compilers, linkers, and packagers to do that work.
- Gradle uses a domain-specific language (DSL) built on top of Groovy to accomplish these tasks.

What Does Android Have To Do with Gradle?

- Google has published the Android Gradle Plugin, which gives Gradle the ability to build Android projects.
- Google is also using Gradle and the Android Gradle Plugin as the build system behind Android Studio.
- The Android Gradle Plugin is updated periodically maintained by Google by listing the Gradle versions supported by each Android Gradle Plugin version
- Gradle can also be installed directly outside of Android Studio
- In case of direct installation define a GRADLE_HOME environment variable, pointing to where you installed Gradle, and to add the bin/ directory inside of Gradle to your PATH environment variable.



The Build Instructions

- The IDE needs to know how to use all components and produce an Android APK file.
- Some details are stored in files that you will edit, by one means or another, from your IDE.
- In Android Studio Gradle uses the information in files named build.gradle to build APKs and other Android outputs.

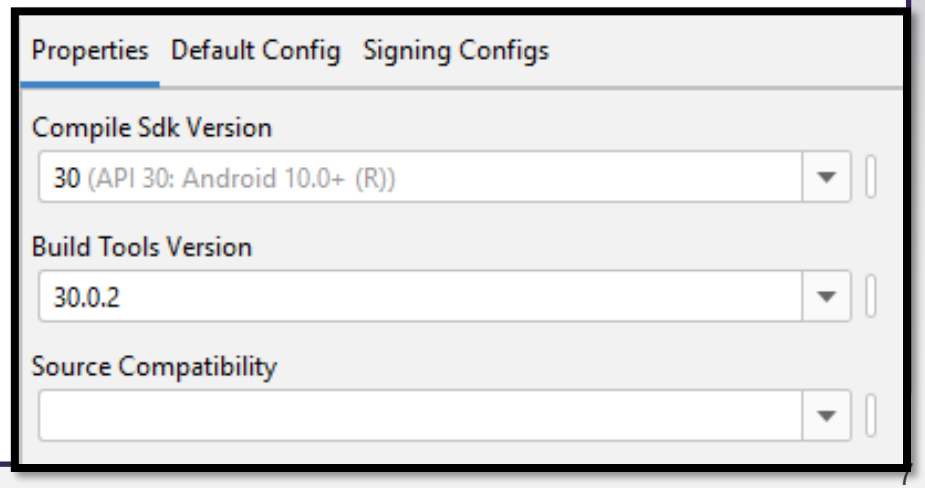
Support Different Platform Versions

- Latest versions of Android provide great APIs but older versions of Android should be also be supported
- It is possible to take advantage of the latest APIs while continuing to support older versions as well.
- Generally, it's a good practice to support about 90% of the active devices, while targeting your app to the latest version.

ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.0 Ice Cream Sandwich	15	
4.1 Jelly Bean	16	99.8%
4.2 Jelly Bean	17	99.2%
4.3 Jelly Bean	18	98.4%
4.4 KitKat	19	98.1%
5.0 Lollipop	21	94.1%
5.1 Lollipop	22	92.3%
6.0 Marshmallow	23	84.9%
7.0 Nougat	24	73.7%
7.1 Nougat	25	66.2%
8.0 Oreo	26	60.8%
8.1 Oreo	27	53.5%
9.0 Pie	28	39.5%
10. Android 10	29	8.2%

compileSdkVersion

- Gradle uses compileSdkVersion of the Android SDK to compile your app
- compileSdkVersion sets the newest APIs available to you
- changing your compileSdkVersion does not change runtime behavior
- compileSdkVersion is not included in your APK: it is purely used at compile time
- You should always compile with the latest SDK
- You'll get all the benefits of new compilation checks on existing code, avoid newly deprecated APIs, and be ready to use new APIs.

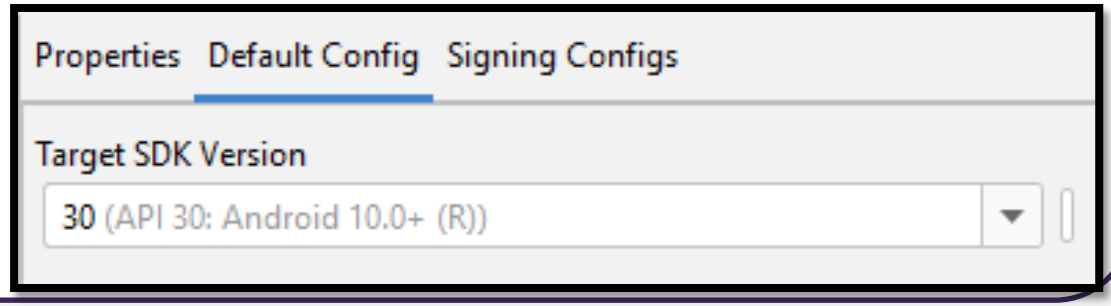


minSdkVersion

- minSdkVersion is the lower bound for your application
- The minSdkVersion is one of the signals the Google Play Store uses to determine which user's devices an app can be installed on.
- During the development by default lint runs against your project, warning you when you use any APIs above your minSdkVersion
- Checking the system version at runtime is a common technique when using APIs only on newer platform versions.
- Of course, if a new API is key to your entire app, then that makes the minSdkVersion choice quite a bit easier.

targetSdkVersion

- It is an integer designating the API Level that the application targets. If not set, the default value equals that given to `minSdkVersion`
- It is the way Android provides forward compatibility by not applying behavior changes unless the targetSdkVersion is updated.
- With some of the behavior changes being very visible to users (the deprecation of the menu button, runtime permissions, etc), updating to target the latest SDK should be a high priority.
- That doesn't mean you have to use every new feature introduced nor should you blindly update your targetSdkVersion without testing — test before updating your targetSdkVersion!



Other Gradle Items of Note

- The android closure has a `compileSdkVersion` property and it also may have a `buildToolsVersion` property.
- `buildToolsVersion` indicates the version of the Android SDK build tools that you wish to use with the project
- If android closure does not have `buildToolsVersion`, the Android Gradle Plugin will use its own default version of these build tools
- The project structure dialog allows to maintain some of Gradle settings.
- For more complex builds we need to work with the Gradle build files directly.

Gradle and SDK versions

- Setting the correct compileSdkVersion, minSdkVersion, and targetSdkVersion is important.
- minSdkVersion and targetSdkVersion also differ from compileSdkVersion in that they are included in your final APK
- Build tools version should match the API you're using. i.e.: if you're using API 30, the build tools' version should be 30.x.x.

```
android {
    compileSdkVersion 30
    buildToolsVersion "30.0.2"

    defaultConfig {
        applicationId "com.nomadlearner.testapp"
        minSdkVersion 23
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"
```

Suggested Values

- The relationship between the three values:

```
minSdkVersion <= targetSdkVersion <= compileSdkVersion
```

- If compileSdkVersion is your ‘maximum’ and minSdkVersion is your ‘minimum’ then your maximum must be at least as high as your minimum and the target must be somewhere in between

- So Ideally,

```
minSdkVersion (lowest possible) <=
    targetSdkVersion == compileSdkVersion (latest SDK)
```

You’ll hit the biggest audience with a low minSdkVersion and look and act the best by targeting and compiling with the latest SDK

Using Build Constants for Version Check

- Android provides a unique code for each platform version in the Build constants class.
- These codes can be used to build conditions that ensure the code that depends on higher API levels is executed only when those APIs are available on the system.

```
private void setUpActionBar() {
    // Make sure we're running on Honeycomb or higher to use ActionBar APIs
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        ActionBar actionBar = getActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
}
```

Use Platform Styles and Themes

- Android provides user experience themes that give apps the look and feel of the underlying operating system.
- These themes can be applied to your app within the manifest file.
- By using these built in styles and themes, your app will naturally follow the latest look and feel of Android with each new release.
- To make your activity look like a dialog box:

```
<activity android:theme="@android:style/Theme.Dialog">
```

- To make your activity have a transparent background:

```
<activity android:theme="@android:style/Theme.Translucent">
```

Use Platform Styles and Themes

- To apply your own custom theme defined in `/res/values/styles.xml`

```
<activity android:theme="@style/CustomTheme">
```

- To apply a theme to your entire app (all activities), add the `android:theme` attribute to the `<application>` element

```
<application android:theme="@style/CustomTheme">
```

Supporting Multiple Screens

- Android devices come with screen sizes ranging from 2.8” tiny smartphones to 46” TVs.
- Android divides these into four buckets, based on physical size and the viewing distance:

1. Small (under 3”)
2. Normal (3” to around 4.5”)
3. Large (4.5” to around 10”)
4. Extra-large (over 10”)

- By default, application will support small and normal screens.
- To support large and extra-large screens <supports-screens> element is added to manifest

```
<supports-screens
  android:largeScreens="true"
  android:normalScreens="true"
  android:smallScreens="false"
  android:xlargeScreens="true" />
```

Blocking Backups

- In manifest the <application> element has an attribute and value like android:allowBackup="true".
- In the short term, change android:allowBackup to be false.

Things In Common Between the Manifest and Gradle

- There are a few key items that can be defined in the manifest and can be overridden in build.gradle statements.

1. Package Name and Application ID (Discussed Already)
2. minSdkVersion and targetSdkVersion (Discussed Already)
3. Version Code and Version Name ???

Package Name and Application ID

- The package also serves as our app's default "application ID". This needs to be a unique identifier, such that:

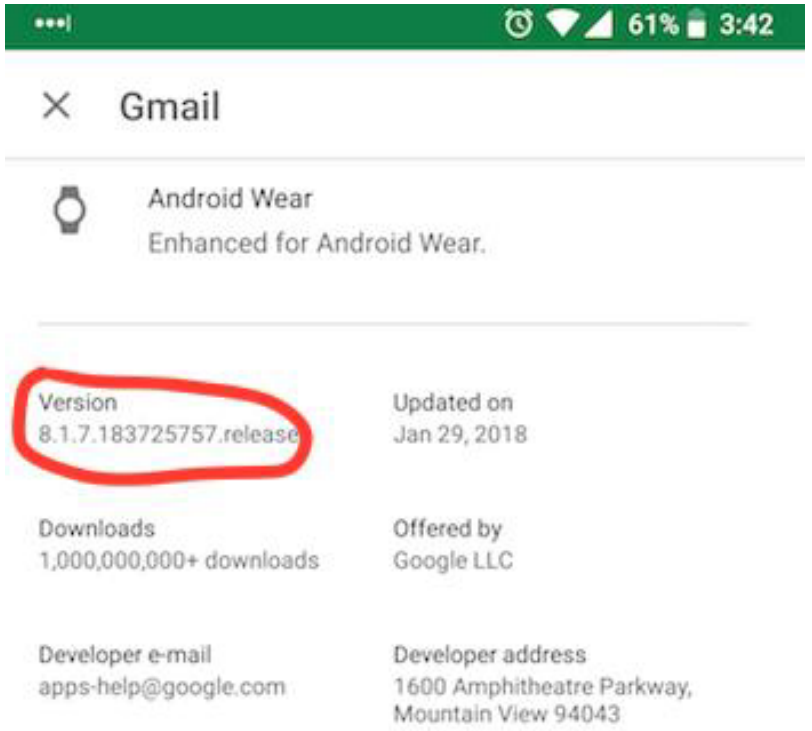
1. no two apps can be installed on the same device at the same time with the same application ID
2. no two apps can be uploaded to the Play Store with the same application ID (and other distribution channels may have the same limitation)

Version Code and Version Name

- The defaultConfig closure has `versionCode` and `versionName` properties.
- They override `android:versionName` and `android:versionCode` attributes on the root <manifest> element in the manifest, though you will not find many projects using those XML attributes.
- Version code and Version name represent the versions of application.
- The `versionName` value is what the user will see for a version indicator in the Applications details screen for your app in their Settings application.
- The version name is used by the Play Store listing also.
- The version name can be any string value you want.

Version Code and Version Name

- The versionCode must be an integer and newer versions must have higher version codes than do older versions.
- Android and the Play Store will compare the version code of a new APK to the version code of an installed application to determine if the new APK is indeed an update.
- The typical approach is to start the version code at 1 and increment it with each production release of your application.
- During development, you can leave these alone, but when you move to production, these attributes will matter greatly.



Key Android Concepts

- In Java entry point into your application was a public static void method named `main()`
- Here instead of `main()`, we create subclasses of some Android-supplied base classes that define various application components.
- In addition, you will create some metadata that tells Android about those subclasses.

Resources

- Resources are static bits of information held outside the Java source code stored as files under the res/ directory
- These are separate from the Java source because they can have multiple definitions to use in different circumstances. (I18N)

String Resources

- Keeping the labels and other bits of text outside the main source code of your application is a good approach. It can help with internationalization (I18N) and localization (L10N)
- It is easier to make corrections if all the strings are in one spot
- If the string value contains a quote or an apostrophe, escape those values with \
- If your message contains [,], or & characters, you will need to use a CDATA section:

```
<string name="report_body">
<![CDATA[
<html>
<body>
<h1>TPS Report for: {{reportDate}}</h1>
<p>Here are the contents of the TPS report:</p>
<p>{{message}}</p>
<p>If you have any questions regarding this report, please
do <b>not</b> ask Mark Murphy.</p>
</body>
</html>
]]>
  </string>
```

String Resources

- For example, a project's strings.xml file could look like this:

```
<resources>
  <string name="app_name">EmPubLite</string>
</resources>
```

- We can reference these string resources from various locations like

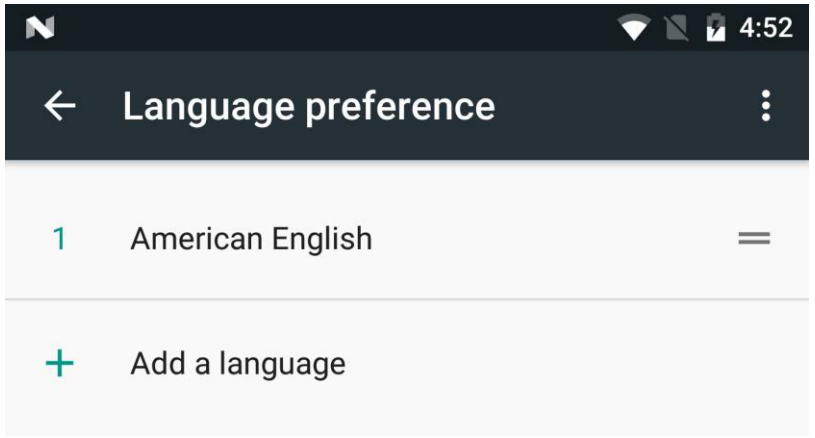
```
<application
  android:allowBackup="false"
  android:icon="@mipmap/ic_launcher"
  android:label="@string/app_name"
```

- The syntax `@string/app_name` tells Android "find the string resource named app_name" by scanning the appropriate strings.xml file (or any other file containing string resources in your res/values/ directory) to try to find app_name.
- String resources support rich text formatting by using lightweight HTML markup: <b>, <i>, and <u>.

```
<resources>
  <string name="b">This has <b>bold</b> in it.</string>
  <string name="i">Whereas this has <i>italics</i>!</string>
</resources>
```

Multi-Locale Support

- Android 7.0+ users can indicate that they support more than one language:



- This has impacts on resource resolution for any locale-dependent resources, such as strings
- Now Android will check multiple languages for resource matches, before falling back to the default language
- You can find out what languages the user has requested via a LocaleList class and its getDefault() static method

Using Picture Resources

- All Android versions support images in the PNG, JPEG, and GIF formats.
- GIF is officially discouraged, however; PNG is the overall preferred format.
- Android also supports some proprietary XML-based image formats
- Many newer versions of Android also support Google's WebP image format, though this is not especially popular.
- Two types of resources that use images : drawables and mipmaps.(nearly identical)
- Mipmaps are used mostly for "launcher icons"
- Drawables hold everything else.

Using Picture Resources

```
<application
    android:allowBackup="false"
    android:icon="@mipmap/ic_launcher"
```

- The manifest simply refers to @mipmap/ic_launcher, telling Android to find a mipmap resource named ic_launcher.
- The resource reference does not indicate the file type of the resource
- The @mipmap/ic_launcher reference does not mention what screen density to use.
- Android will choose the right screen density to use, based upon the device that is running your app.
- If Android detects that the device has a screen density for which you lack an icon, Android will take the next-closest one and scale it.

Picture Resources

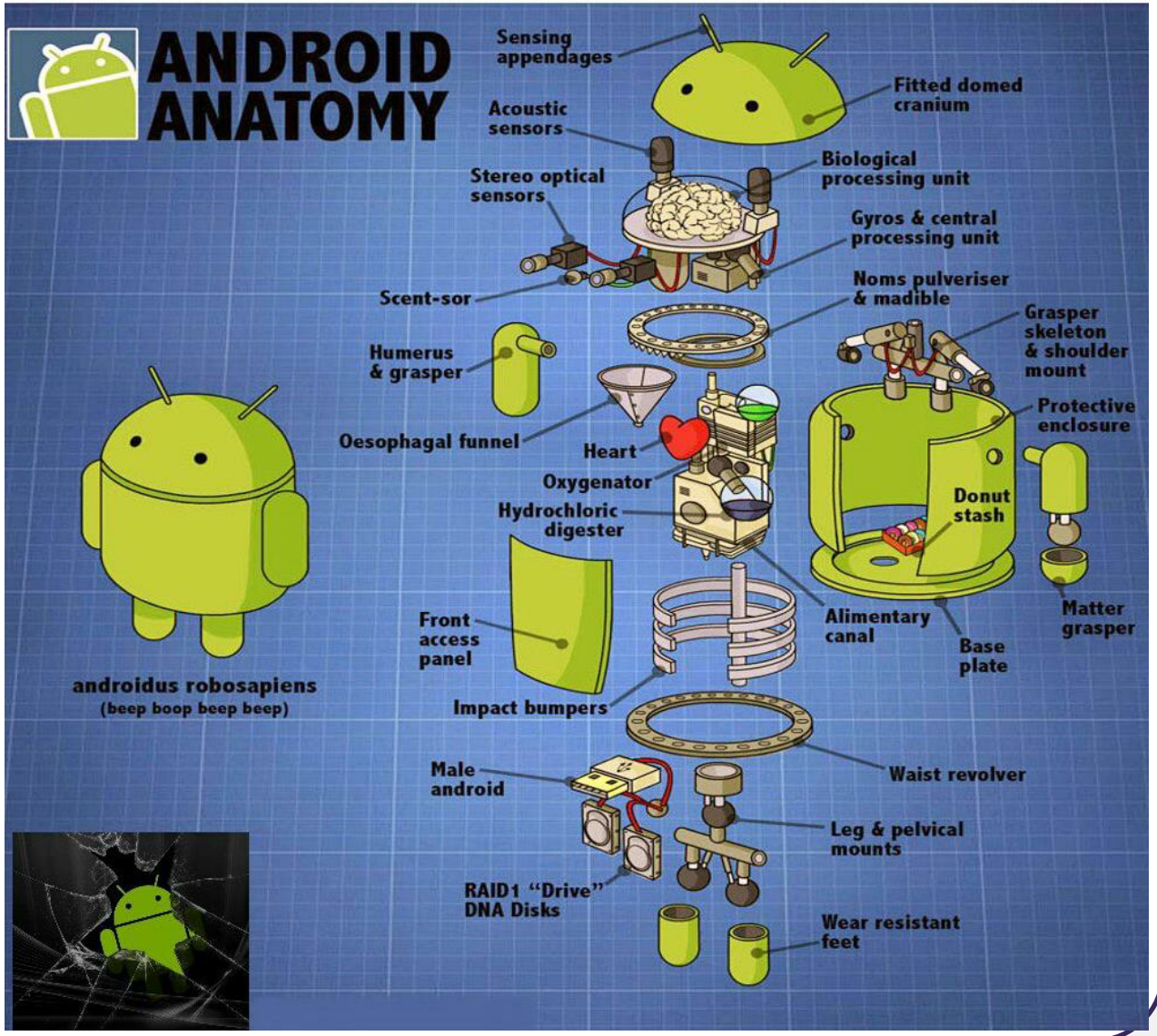
Read more about Picture Resources at

Page # 102 to 107, Some Words About Resources from, The Busy Coder's Guide to Android Development by Mark L. Murphy, CommonsWare publications

The Anatomy of an Android Application

- Various components that are used to construct an application may include:

1. Android Activities
2. Android Fragments
3. Android Intents
4. Broadcast Intents
5. Broadcast Receivers
6. Android Services
7. Content Providers
8. The Application Manifest
9. Application Resources
10. Application Context

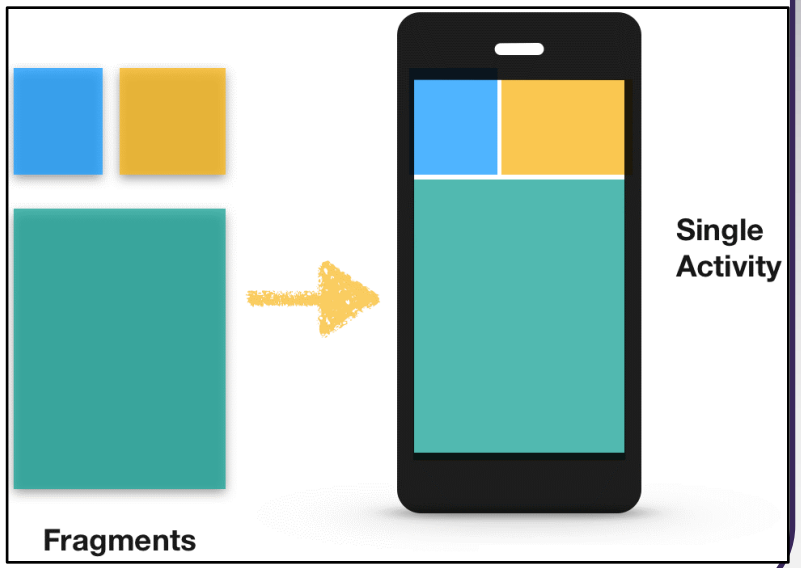


1. Android Activities

- The building block of the user interface is the activity.
- You can think of an activity as being the Android equivalent for the window or dialog in a desktop application, or the page in a classic Web app.
- It represents a chunk of your user interface and, in some cases, a discrete entry point into your app (i.e., a way for other apps to link to your app).
- Normally, an activity will take up most of the screen, leaving space for some “chrome” bits like the clock, signal strength indicators, and so forth.
- Activities are short-lived and can be shut down at any time, such as when the user presses the BACK button.

2. Android Fragments

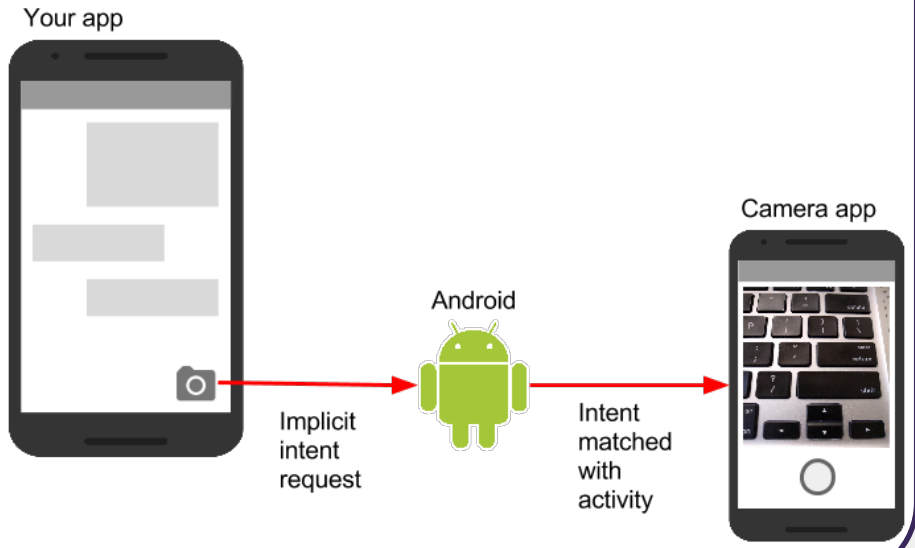
- An activity represents a single user interface screen within an app.
- One alternative is to break the activity into different sections.
- Each of these sections is referred to as a fragment, each of which consists of part of the user interface layout and a matching class file (declared as a subclass of the Android Fragment class).
- In this scenario, an activity simply becomes a container into which one or more fragments are embedded.
- Fragments provide an efficient alternative to having each user interface screen represented by a separate activity.
- Instead, an app can consist of a single activity that switches between different fragments, each representing a different app screen.



3. Android Intents

- Intents are the mechanism by which one activity can launch another and implement the flow through the activities that make up an application.
- Intents consist of a description of the operation to be performed and, optionally, the data on which it is to be performed.
- Explicit intents can request the launch of a specific activity by referencing the activity by class name

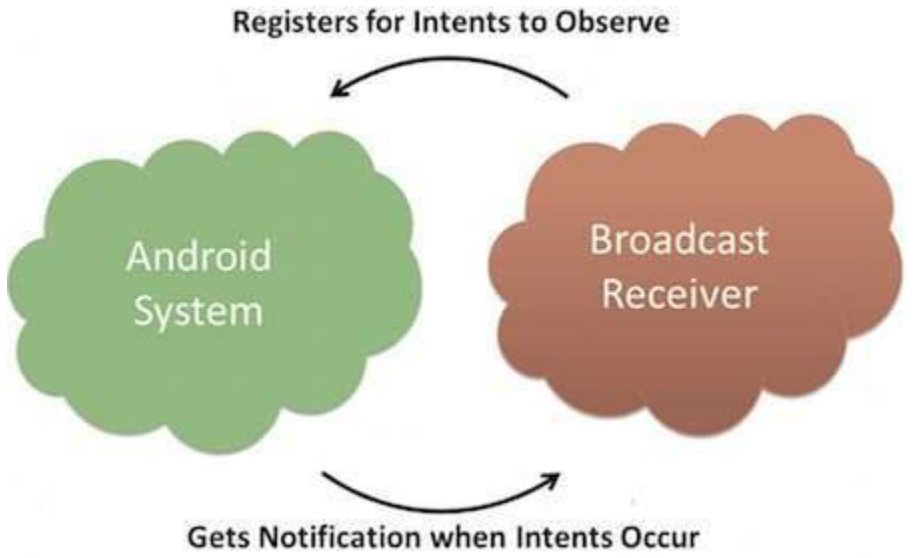
- Implicit intents can request the launch by stating either the type of action to be performed or providing data of a specific type on which the action is to be performed.
- In the case of implicit intents, the Android runtime will select the activity to launch that most closely matches the criteria specified by the Intent using a process referred to as Intent Resolution.



4. Broadcast Intents

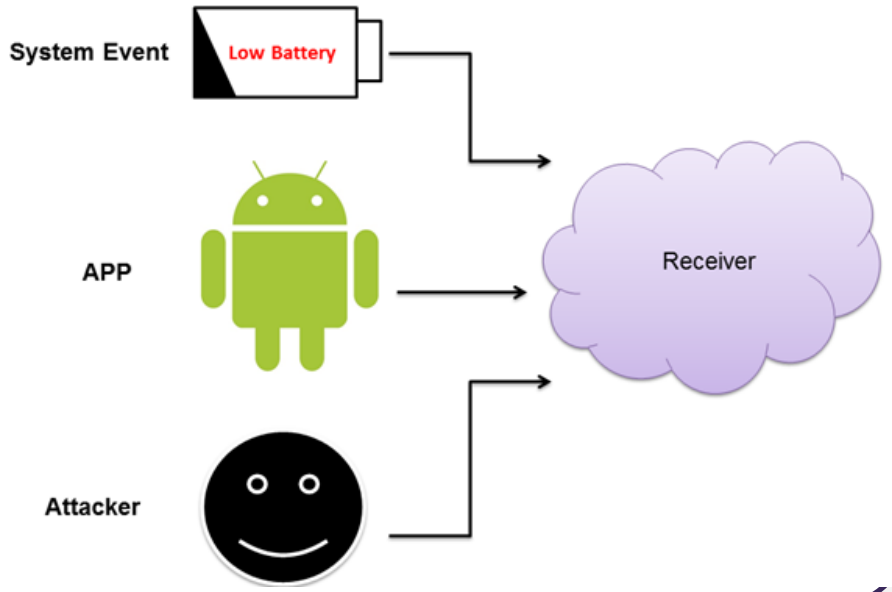
- Broadcast Intent, is a system wide intent that is sent out to all applications that have registered an “interested” Broadcast Receiver.
- For example, the Android system send out Broadcast Intents to indicate changes in device status like completion of system start up, connection of an external power source to the device or the screen being turned on or off .

- Normal (asynchronous) Broadcast Intent is sent to all interested Broadcast Receivers at the same time,
- Ordered Broadcast Intent is sent to one receiver at a time where it can be processed and then either aborted or allowed to be passed to the next Broadcast Receiver.



5. Broadcast Receivers

- Broadcast Receivers are the mechanism by which applications can respond to Broadcast Intents.
- A Broadcast Receiver must be registered by an application and configured with an Intent Filter to indicate the types of broadcast in which it is interested.
- When a matching intent is broadcast, the receiver will be invoked by the Android runtime regardless of whether the application that registered the receiver is currently running.
- The receiver then has 5 seconds in which to complete any tasks required of it (such as launching a Service, making data updates or issuing a notification to the user) before returning.
- Broadcast Receivers operate in the background and do not have a user interface.

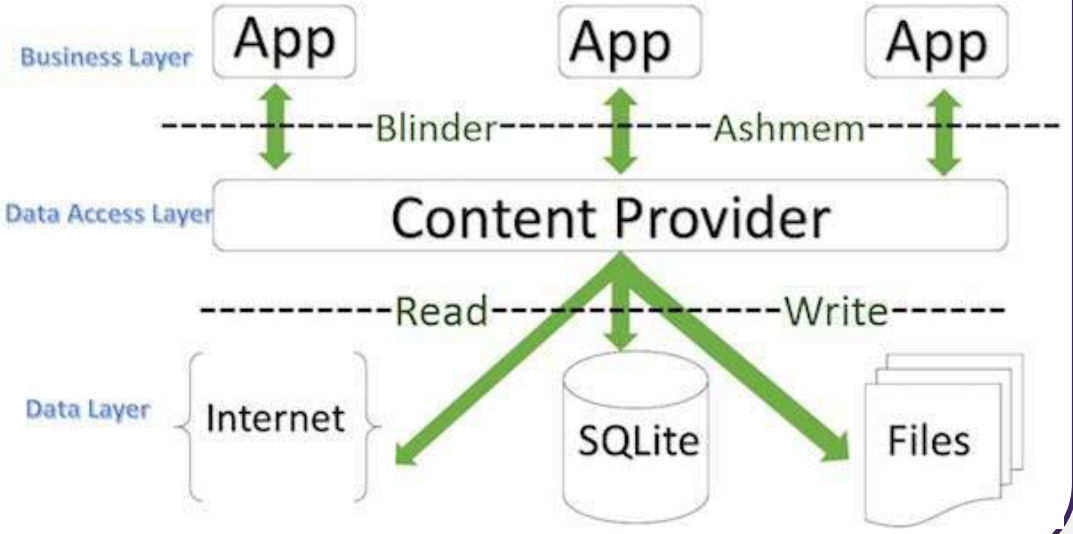


6. Android Services

- Services are designed to keep running, if needed, independent of any activity, for a moderate period of time.
- You might use a service for checking for updates to an RSS feed, or to play back music even if the controlling activity is no longer operating.
- You will also use services for scheduled tasks and for exposing custom APIs to other applications on the device.
- Although Services lack a user interface, they can still notify the user of events using notifications and toasts and are also able to issue Intents.

7. Content Providers

- Content Providers implement a mechanism for the sharing of data between applications.
- Any application can provide other applications with access to its underlying data through the implementation of a Content Provider (add, remove and query data with permissions)
- Access to the data is provided via a Universal Resource Identifier (URI) defined by the Content Provider.
- Data can be shared in the form of a file or an entire SQLite database.
- The native Android applications include several standard Content Providers allowing applications to access data such as contacts and media files

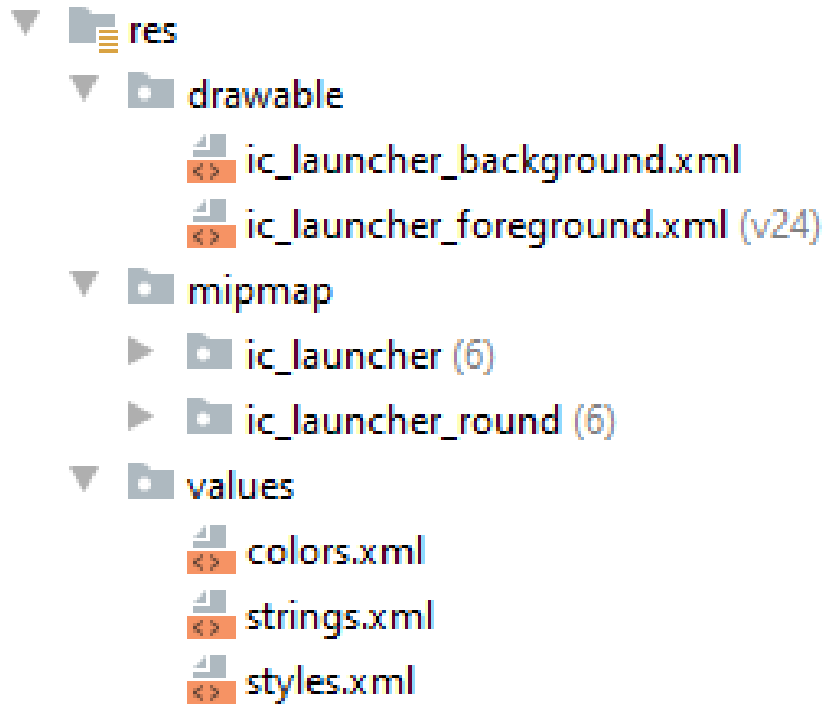


8. The Application Manifest

- The glue that pulls together the various elements that comprise an application is the Application Manifest file.
- It is within this XML based file that the application outlines the activities, services, broadcast receivers, data providers and permissions that make up the complete application.

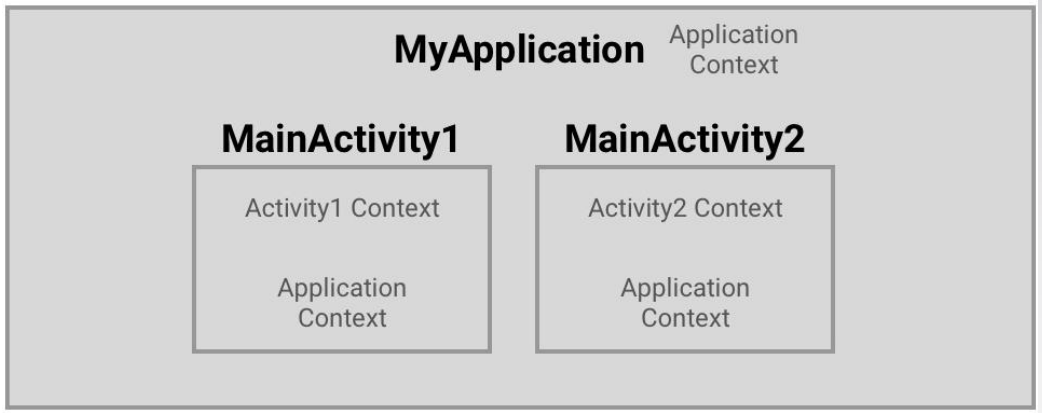
9. Application Resources

- Resource files contain resources such as the strings, images, fonts and colors that appear in the user interface together with the XML representation of the user interface layouts.
- By default, these files are stored in the `/res` sub-directory of the application project's hierarchy.
- An Android application package (apk) contains manifest file, the Dex files that contain the byte code and a collection of resource files.



10. Application Context

- When an application is compiled, a class named R is created that contains references to the application resources.
- The application manifest file and these resources combine to create what is known as the Application Context.
- Application context is represented by the Android Context class. It can be used to gain access to the application resources at runtime.
- Various methods can be called on an application's context to gather information and make changes to the application's environment at runtime.



Recommended Readings

- Picking your compileSdkVersion, minSdkVersion, and targetSdkVersion at https://medium.com/androiddevelopers/picking-your-compilesdkversion-minsdkversion-targetsdkversion-a098a0341ebd
- User Guide: developer.android.com/studio/intro
- Page # 01 to 06, Key Android Concepts from, The Busy Coder's Guide to Android Development by Mark L. Murphy, CommonsWare publications
- Page # 75 to 90, Introducing Gradle and the Manifest from, The Busy Coder's Guide to Android Development by Mark L. Murphy, CommonsWare publications
- Page # 91 to 94, Tutorial #3 - Manifest Changes from, The Busy Coder's Guide to Android Development by Mark L. Murphy, CommonsWare publications
- Page # 95 to 100, Some Words About Resources from, The Busy Coder's Guide to Android Development by Mark L. Murphy, CommonsWare publications
- Page # 77 to 80, Chapter # 10: The Anatomy of an Android Application from Android Studio 4.0 Development Essentials by Neil Smyth, Payload Media, Inc.