CHECH TALK #3

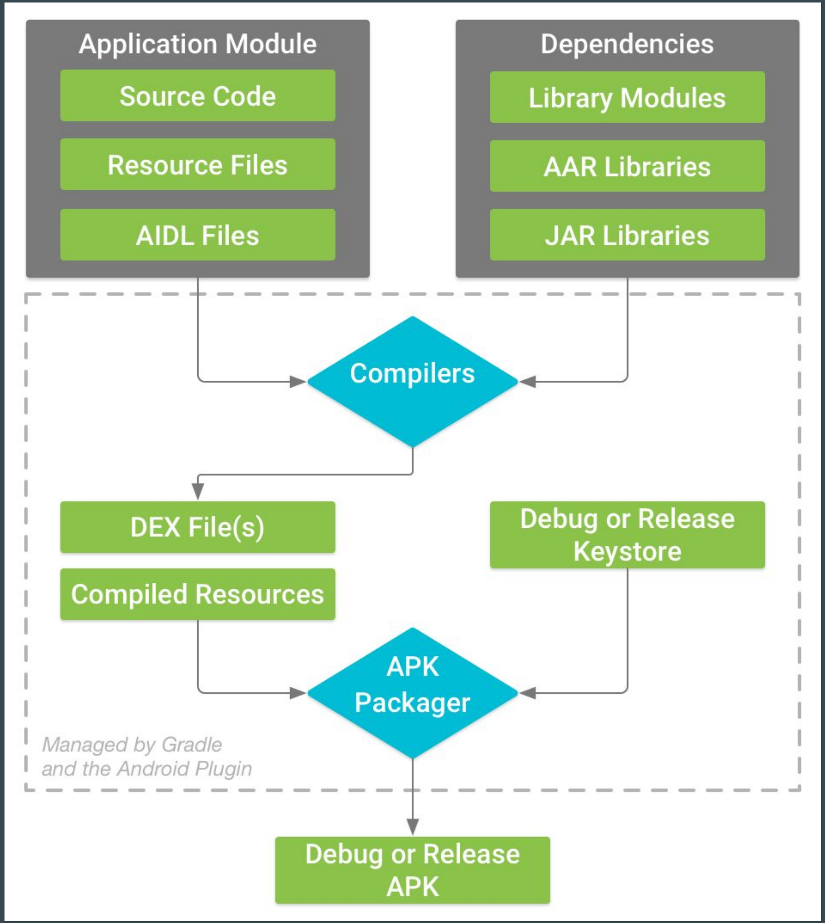
android

- Application Fundamentals
    - The Build Process
- App Components
    - The Manifest File
    - Declaring components
    - Declaring components capabilities
    - Declaring app requirements
    - Using permissions
- App Resources
- Intents and Intents Filters
    - Building an Intent
- Activities
- UI Overview
    - Input Controls
    - Layouts
- Support Library

Application Fundamentals

- Android apps are written in the Java programming language.
- The Android SDK tools compile your code (along with any data and resource files) into an .apk file
- Once installed, each Android app lives in its own security sandbox:
    - The OS is a multi-user Linux system in which each app is a different user.
    - Each process has its own virtual machine (Dalvik, ART), so an app's code runs in isolation from other apps.
- Android apps are created using Android Studio (based on IntelliJ IDEA)
- Android apps use Gradle as the default build system.

The Build Process



App Components

There are four different types of app components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

- Activities
  - An *`Activity`* represents a single screen with a user interface.
- Services
  - A *`Service`* is a component that runs in the background to perform long-running operations. It does not provide a user interface.
- Content Providers
  - A *`ContentProvider`* manages a shared set of app data.
- Broadcast Receivers
  - A *`BroadcastReceiver`* is a component that responds to system-wide broadcast announcements.

App Components

A unique aspect of the Android system design is that any app can start another app's component. Because the system runs each app in a separate process with file permissions that restrict access to other apps, your app cannot directly activate a component from another app, you must deliver a message to the system that specifies your *`Intent`** to start a particular component.

* more on this later

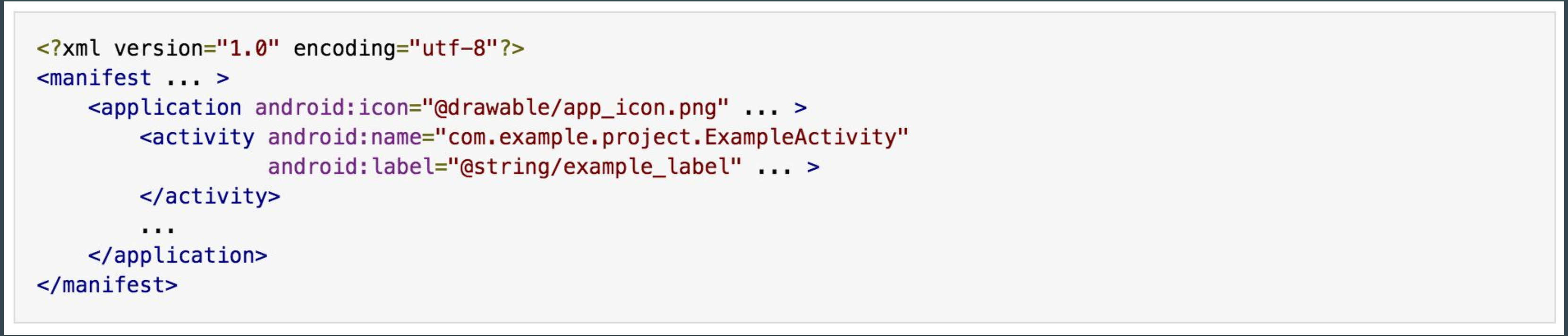
The Manifest File

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. This file:

- declare the app's components
- identify any user permissions the app required
- declare the minimum API level
- declare hardware and software features used or required by the app
- and more...

Declaring components

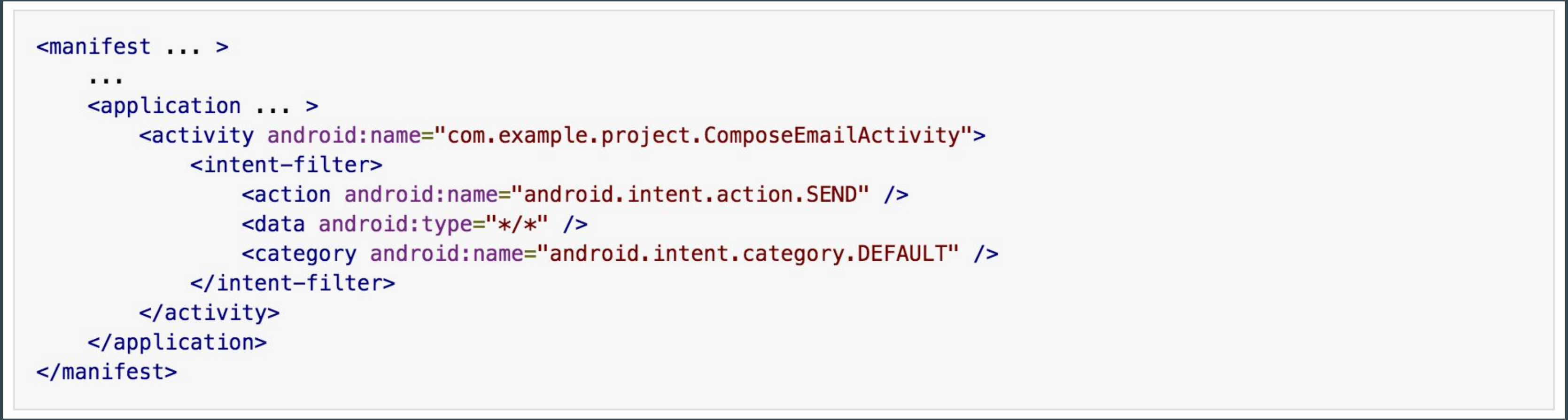
The primary task of the manifest is to inform the system about the app's components. For example, a manifest file can declare an `Activity` as follows:



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

Declaring component capabilities

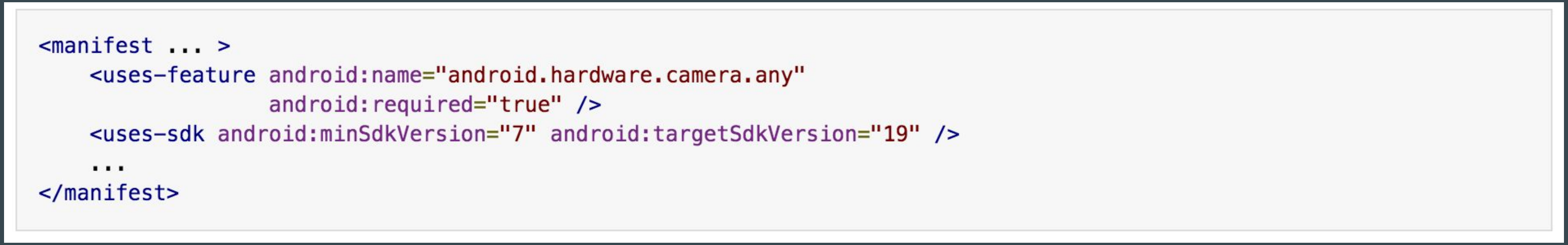
When you declare an `Activity` in your app's manifest, you can optionally include intent filters that declare the capabilities of the activity so it can respond to intents from other apps. You can declare an intent filter for your component by adding an `<intent-filter>` element as a child of the component's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



```xml
<manifest ... >
    ...
    <application ... >
        <activity android:name="com.example.project.ComposeEmailActivity">
            <intent-filter>
                <action android:name="android.intent.action.SEND" />
                <data android:type="*/*" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Declaring app requirements

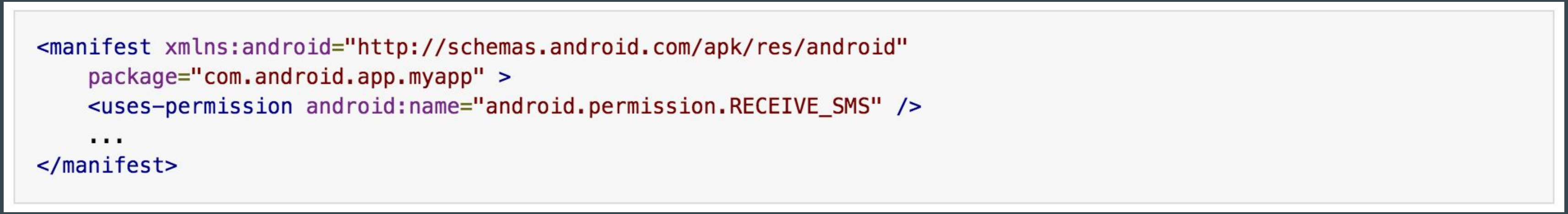
There are a variety of devices powered by Android and not all of them provide the same features and capabilities. In order to prevent your app from being installed on devices that lack features needed by your app, it's important that you clearly define a profile for the types of devices your app supports by declaring device and software requirements in your manifest file.



```xml
<manifest ... >
    <uses-feature android:name="android.hardware.camera.any"
                  android:required="true" />
    <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="19" />
    ...
</manifest>
```

Using Permissions

A basic Android application has no permissions associated with it by default, meaning it cannot do anything that would adversely impact the user experience or any data on the device. To make use of protected features of the device, you must include one or more *`<uses-permission>`* tags in your app manifest.

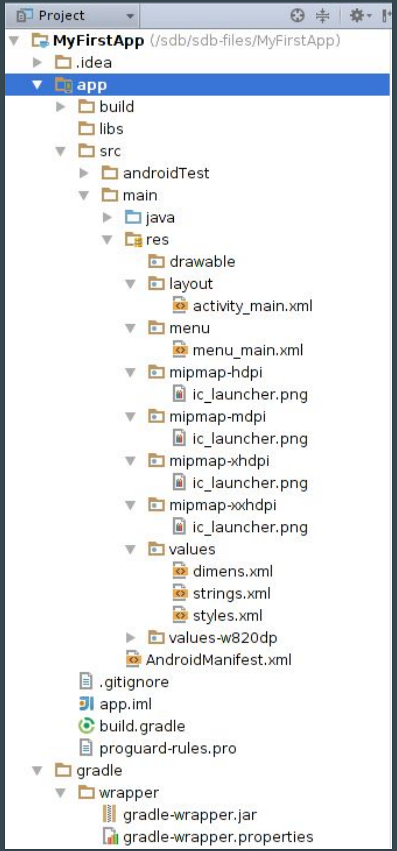


```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.app.myapp" >
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    ...
</manifest>
```

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.

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.



```
Project
MyFirstApp (/sdb/sdb-files/MyFirstApp)
  .idea
  app
    build
    libs
    src
      androidTest
      main
        java
        res
          drawable
          layout
            activity_main.xml
          menu
            menu_main.xml
          mipmap-hdpi
            ic_launcher.png
          mipmap-mdpi
            ic_launcher.png
          mipmap-xhdpi
            ic_launcher.png
          mipmap-xxhdpi
            ic_launcher.png
          values
            dimens.xml
            strings.xml
            styles.xml
          values-w820dp
        AndroidManifest.xml
    .gitignore
    app.iml
    build.gradle
    proguard-rules.pro
  gradle
    wrapper
      gradle-wrapper.jar
      gradle-wrapper.properties
```

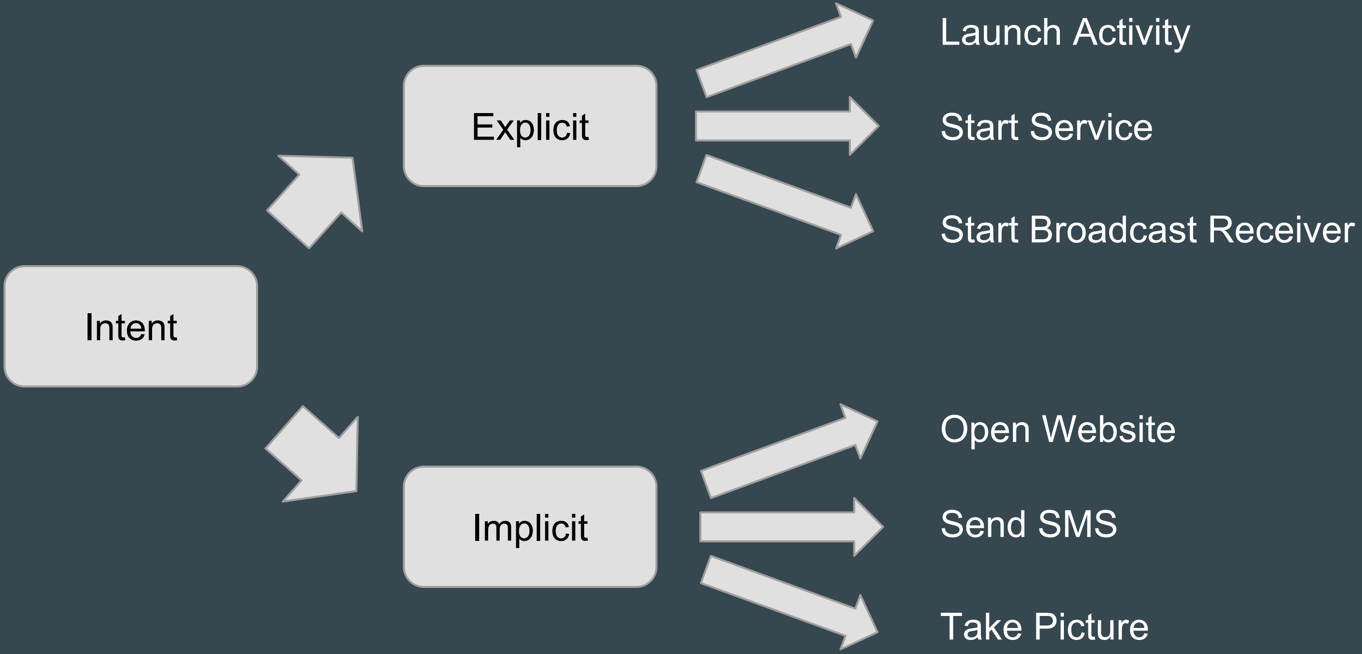
Intents and Intent Filters

An `Intent` is a messaging object you can use to request an action from another app component.

There are two types of intents:

- **Explicit intents** specify the component to start by name (the fully-qualified class name). You'll typically use an explicit intent to start a component in your own app.
- **Implicit intents** do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.

Intents and Intent Filters



Building an Intent

- Component name
  - The name of the component to start. This is optional, but it's the critical piece of information that makes an intent explicit. Without a component name, the intent is implicit.
- Action
  - A string that specifies the generic action to perform (such as ACTION_VIEW, ACTION_SEARCH or ACTION_CALL).
- Data
  - The URI that references the data to be acted on and/or the MIME type of that data. The type of data supplied is generally dictated by the intent's action. (e.g. "image/jpeg")
- Extras
  - Key-value pairs that carry additional information required to accomplish the requested action.
- Flags
  - Flags defined in the Intent class that function as metadata for the intent.

Building an Intent

- Example explicit *`Intent`*

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

- Example implicit *`Intent`*

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

Activities

An Activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map. Each activity is given a window in which to draw its user interface.

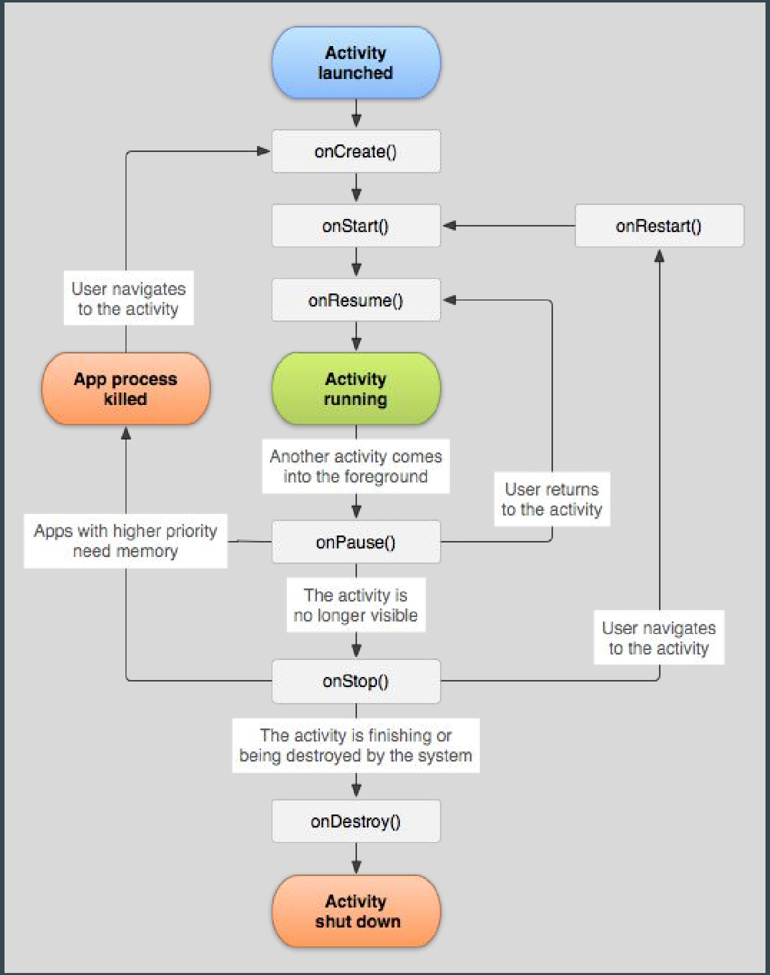
An activity can exist in essentially three states:

- Resumed
- Paused
- Stopped

If an activity is paused or stopped, the system can drop it from memory either by asking it to finish (calling its finish() method), or simply killing its process.

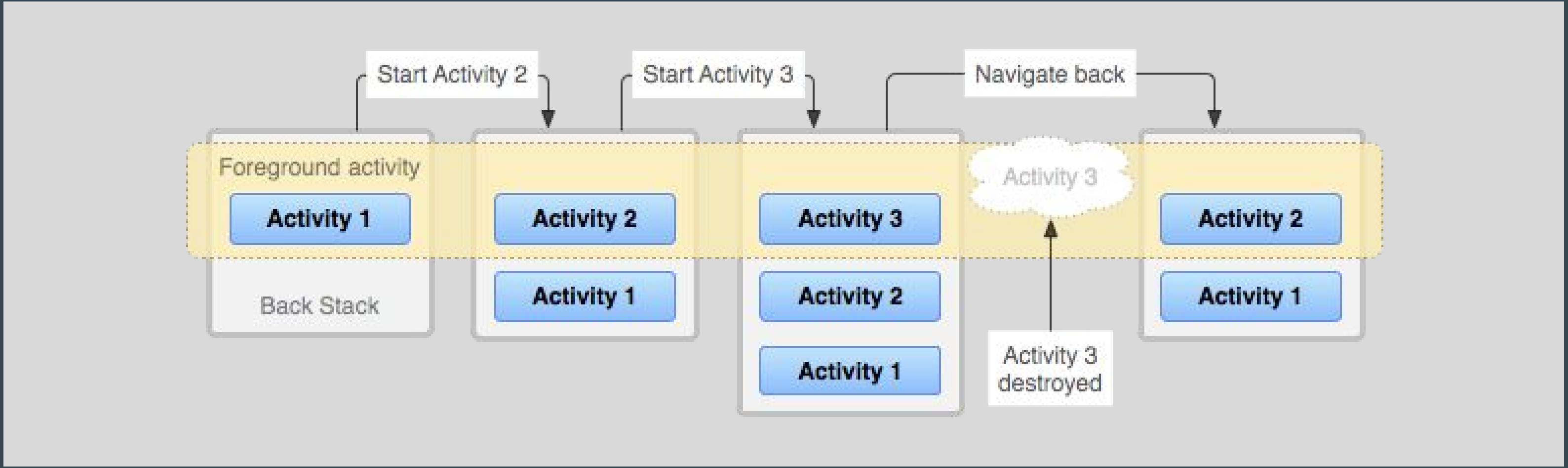
Activities

When an activity is stopped, it is notified of this change in state through the activity's lifecycle callback methods. Each callback provides you the opportunity to perform specific work that's appropriate to that state change.


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

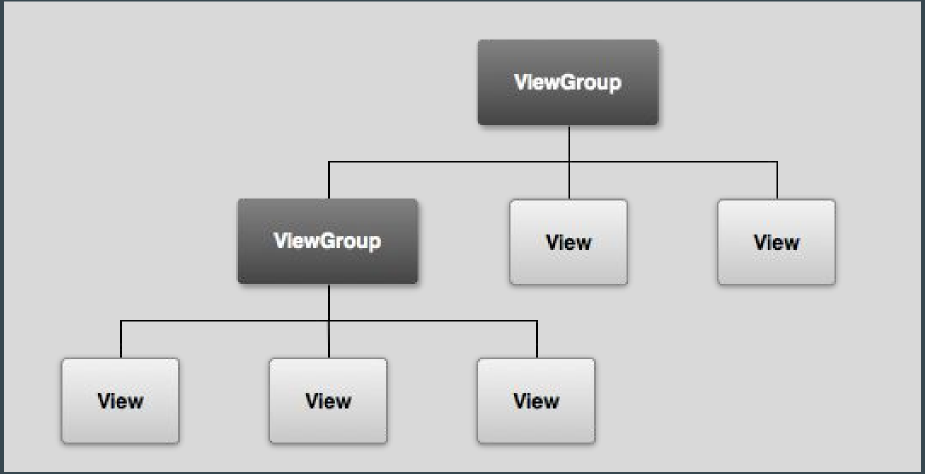
Activities

An application usually consists of multiple activities that are loosely bound to each other. Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the "back stack").



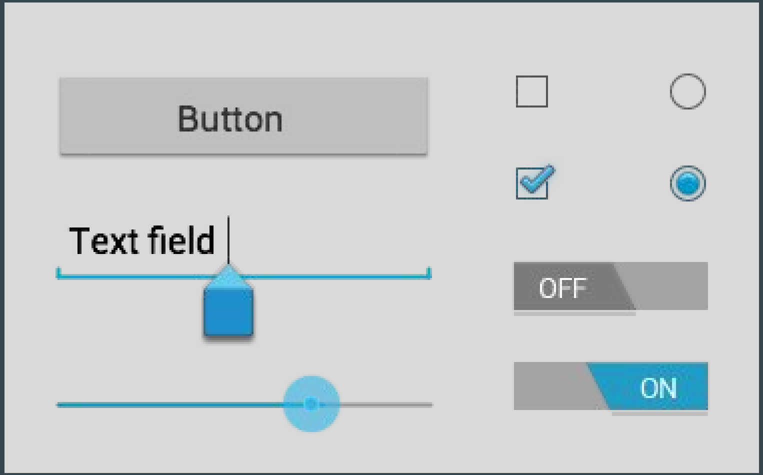
UI Overview

All user interface elements in an Android app are built using `View` and `ViewGroup` objects. A `View` is an object that draws something on the screen that the user can interact with. A `ViewGroup` is an object that holds other `View` (and `ViewGroup`) objects in order to define the layout of the interface.



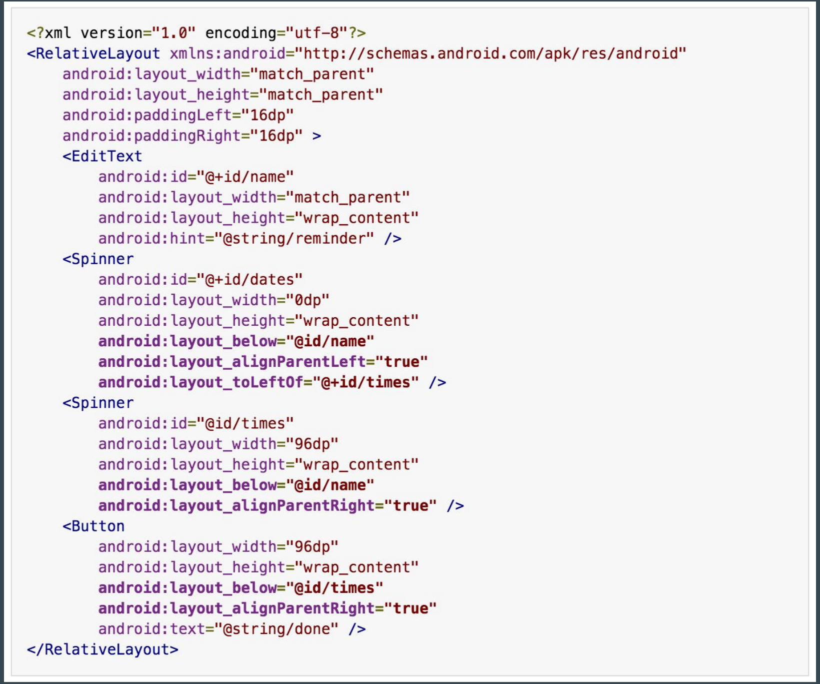
Input Controls

- Button
- EditText, AutoCompleteEditText
- CheckBox
- RadioGroup, RadioButton
- ToggleButton
- Spinner
- DatePicker, TimePicker

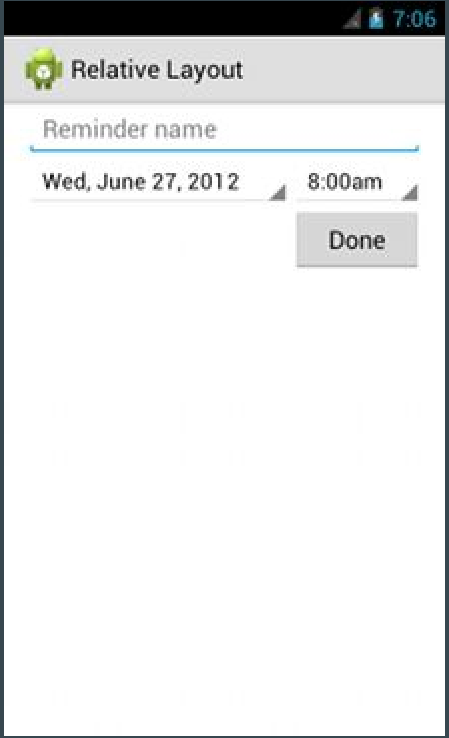


Layouts

A layout defines the visual structure for a user interface.



```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```



Support Library

The Android Support Library offers a number of features that are not built into the framework. These libraries offer backward-compatible versions of new features, provide useful UI elements that are not included in the framework, and provide a range of utilities that apps can draw on.

- *`com.android.support:support-fragment:24.2.0`*
- *`com.android.support:recyclerview-v7:24.2.0`*
- *`com.android.support:design:24.2.0`*

Questions?