**3. Mobile Application Development**

**(Android Platform)**

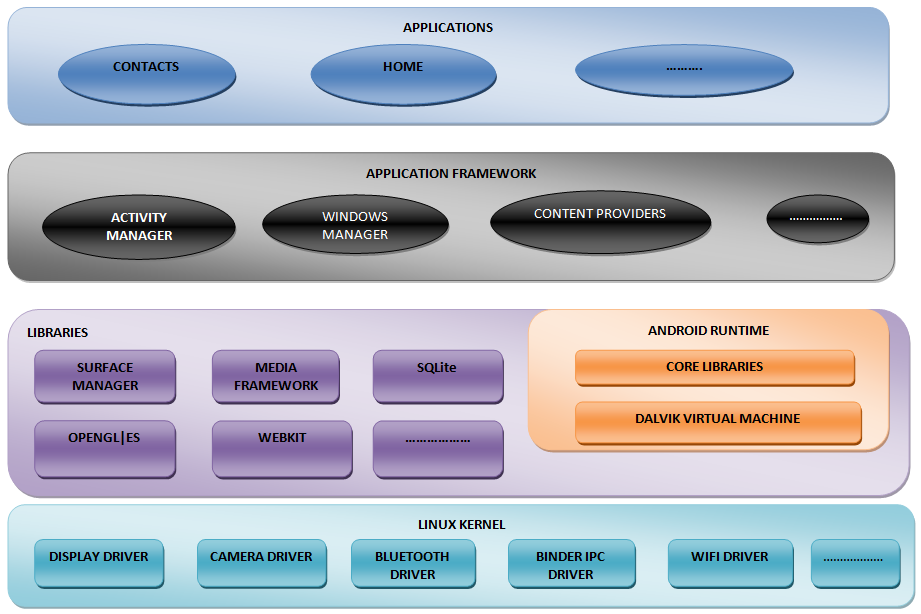
**Android – Architecture & Components**

Android operating system is a stack of software components which is roughly divided into

* Five sections and
* Four main layers

As shown below in the architecture diagram.





## Linux kernel

At the bottom of the layers is Linux - Linux 3.6 with approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

## Libraries

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, **SQLite** database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

**SQLite** is a [relational database management system](https://en.wikipedia.org/wiki/Relational_database_management_system) contained in a C programming [library](https://en.wikipedia.org/wiki/Library_(computer_science)). In contrast to many other database management systems, SQLite is not a [client–server](https://en.wikipedia.org/wiki/Client%E2%80%93server) database engine. Rather, it is embedded into the end program.

## Android Libraries

This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access. A summary of some key core Android libraries available to the Android developer is as follows −

* **android.app** − Provides access to the application model and is the cornerstone of all Android applications.
* **android.content** − Facilitates content access, publishing and messaging between applications and application components.
* **android.database** − Used to access data published by content providers and includes SQLite database management classes.
* **android.opengl** − A Java interface to the OpenGL ES 3D graphics rendering API.
* **android.os** − Provides applications with access to standard operating system services including messages, system services and inter-process communication.
* **android.text** − Used to render and manipulate text on a device display.
* **android.view** − The fundamental building blocks of application user interfaces.
* **android.widget** − A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
* **android.webkit** − A set of classes intended to allow web-browsing capabilities to be built into applications.

Having covered the Java-based core libraries in the Android runtime, it is now time to turn our attention to the C/C++ based libraries contained in this layer of the Android software stack.

## Android Runtime

This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called **Dalvik Virtual Machine** which is a kind of Java Virtual Machine specially designed and optimized for Android.

The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.

The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

## Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

The Android framework includes the following key services –

* **Activity Manager** − Controls all aspects of the application lifecycle and activity stack.
* **Content Providers** − Allows applications to publish and share data with other applications.
* **Resource Manager** − Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
* **Notifications Manager** − Allows applications to display alerts and notifications to the user.
* **View System** − An extensible set of views used to create application user interfaces.

## Applications

We will find all the Android application at the top layer. We will write our application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games etc.

# Android - Application Components

Application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest file *AndroidManifest.xml* that describes each component of the application and how they interact.

There are following four main components that can be used within an Android application –

|  |  |
| --- | --- |
| **Sr.No** | **Components & Description** |
| 1 | **Activities**  They dictate the UI and handle the user interaction to the smart phone screen. |
| 2 | **Services**  They handle background processing associated with an application. |
| 3 | **Broadcast Receivers**  They handle communication between Android OS and applications. |
| 4 | **Content Providers**  They handle data and database management issues. |

## Activities

An activity represents a single screen with a user interface, in-short Activity performs actions on the screen. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.

An activity is implemented as a subclass of **Activity** class as follows –

public class MainActivity extends Activity {

}

## Services

A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.

A service is implemented as a subclass of **Service** class as follows −

public class MyService extends Service {

}

## Broadcast Receivers

Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and each message is broadcaster as an **Intent** object.

public class MyReceiver extends BroadcastReceiver {

public void onReceive(context,intent){}

}

## Content Providers

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the *ContentResolver* class. The data may be stored in the file system, the database or somewhere else entirely.

A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

public class MyContentProvider extends ContentProvider {

public void onCreate(){}

}

## Additional Components

There are additional components which will be used in the construction of above mentioned entities, their logic, and wiring between them. These components are –

|  |  |
| --- | --- |
| **S.No** | **Components & Description** |
| 1 | **Fragments**  Represents a portion of user interface in an Activity. |
| 2 | **Views**  UI elements that are drawn on-screen including buttons, lists forms etc. |
| 3 | **Layouts**  View hierarchies that control screen format and appearance of the views. |
| 4 | **Intents**  Messages wiring components together. |
| 5 | **Resources**  External elements, such as strings, constants and drawable pictures. |
| 6 | **Manifest**  Configuration file for the application. |

# Android - Hello World Example

Let us start actual programming with Android Framework. Before we start writing our first example using Android SDK, we have to make sure that we have set-up our Android development environment properly as explained in [Android - Environment Set-up](https://www.tutorialspoint.com/android/android_environment_setup.htm). Also we should have a little bit working knowledge with Android studio.

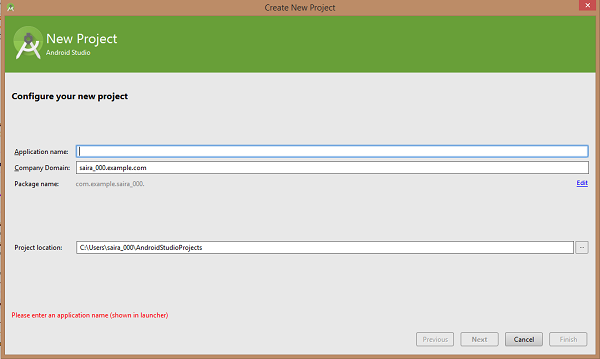
So let us proceed to write a simple Android Application which will print "Hello World!".

## Create Android Application

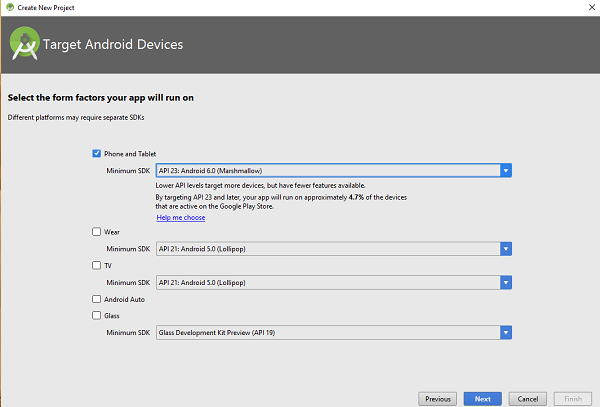
The first step is to create a simple Android Application using Android studio. When we click on Android studio icon, it will show screen as shown below



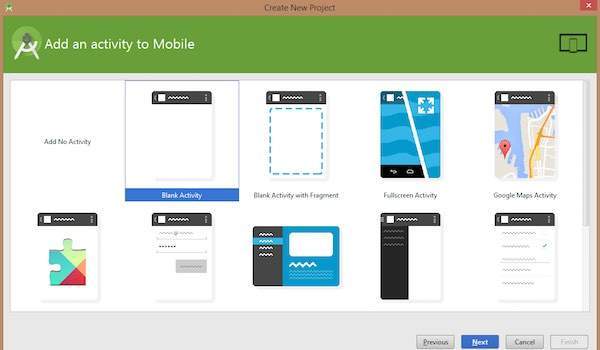
We can start our application development by calling start a new android studio project. In a new installation frame should ask Application name, package information and location of the project.−



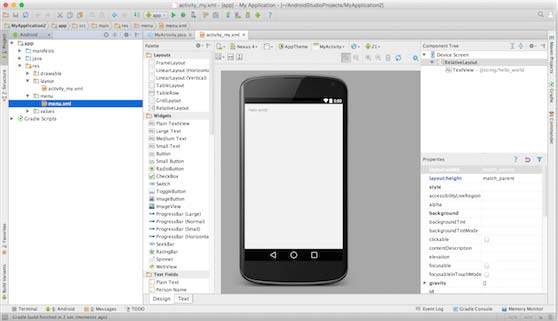
After entered application name, it going to be called select the form factors our application runs on, here need to specify Minimum SDK, API23: Android 6.0(Mashmallow) –



The next level of installation should contain selecting the activity to mobile, it specifies the default layout for Applications.

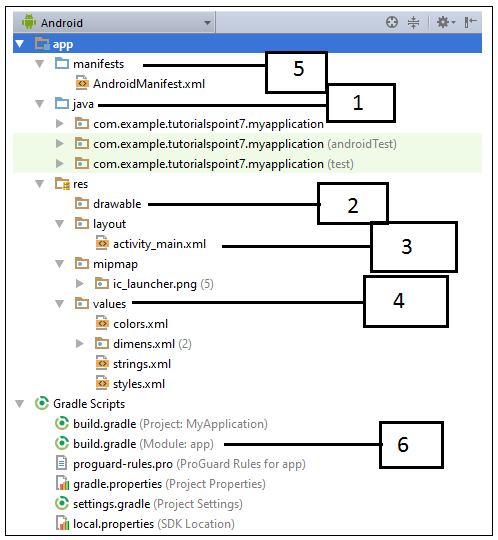


At the final stage it going to be open development tool to write the application code.



## Anatomy of Android Application

Before we run our app, we should be aware of a few directories and files in the Android project –



|  |  |
| --- | --- |
| **Sr.No.** | **Folder, File & Description** |
| 1 | **Java**  This contains the **.java** source files for our project. By default, it includes an *MainActivity.java* source file having an activity class that runs when our app is launched using the app icon. |
|  |  |
| 2 | **res/drawable-hdpi**  This is a directory for drawable objects that are designed for high-density screens. |
| 3 | **res/layout**  This is a directory for files that define our app's user interface. |
| 4 | **res/values**  This is a directory for other various XML files that contain a collection of resources, such as strings and colours definitions. |
| 5 | **AndroidManifest.xml**  This is the manifest file which describes the fundamental characteristics of the app and defines each of its components. |
| 6 | **Build.gradle**  This is an auto generated file which contains compileSdkVersion, buildToolsVersion, applicationId, minSdkVersion, targetSdkVersion, versionCode and versionName |

Following section will give a brief overview few of the important application files.

## The Main Activity File

The main activity code is a Java file **MainActivity.java**. This is the actual application file which ultimately gets converted to a Dalvik executable and runs your application. Following is the default code generated by the application wizard for *Hello World!* application –

package com.example.helloworld;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

}

Here, *R.layout.activity\_main* refers to the *activity\_main.xml* file located in the *res/layout* folder. The *onCreate()* method is one of many methods that are figured when an activity is loaded.

## The Manifest File

Whatever component we develop as a part of our application, we must declare all its components in a *manifest.xml* which resides at the root of the application project directory. This file works as an interface between Android OS and our application, so if we do not declare our component in this file, then it will not be considered by the OS. For example, a default manifest file will look like as following file –

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.tutorialspoint7.myapplication">

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<activity android:name=".MainActivity">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>

Here <application>...</application> tags enclosed the components related to the application. Attribute *android:icon* will point to the application icon available under *res/drawable-hdpi*. The application uses the image named ic\_launcher.png located in the drawable folders.

The <activity> tag is used to specify an activity and  *android:name*  attribute specifies the fully qualified class name of the *Activity* subclass and the *android:label* attributes specifies a string to use as the label for the activity. We can specify multiple activities using <activity> tags.

The **action** for the intent filter is named *android.intent.action.MAIN* to indicate that this activity serves as the entry point for the application.

The **category** for the intent-filter is named  *android.intent.category.LAUNCHER* to indicate that the application can be launched from the device's launcher icon.

The *@string* refers to the *strings.xml* file explained below. Hence, *@string/app\_name* refers to the *app\_name* string defined in the strings.xml file, which is "Hello World". Similar way, other strings get populated in the application.

Following is the list of tags which we will use in our manifest file to specify different Android application components –

* <activity>elements for activities
* <service> elements for services
* <receiver> elements for broadcast receivers
* <provider> elements for content providers

## The Strings File

The **strings.xml** file is located in the *res/values* folder and it contains all the text that our application uses. For example, the names of buttons, labels, default text, and similar types of strings go into this file. This file is responsible for their textual content. For example, a default strings file will look like as following file –

<resources>

<string name="app\_name">HelloWorld</string>

<string name="hello\_world">Hello world!</string>

<string name="menu\_settings">Settings</string>

<string name="title\_activity\_main">MainActivity</string>

</resources>

## The Layout File

The **activity\_main.xml** is a layout file available in *res/layout* directory, that is referenced by our application when building its interface. We will modify this file very frequently to change the layout of our application. For our "Hello World!" application, this file will have following content related to default layout –

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" >

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_centerHorizontal="true"

android:layout\_centerVertical="true"

android:padding="@dimen/padding\_medium"

android:text="@string/hello\_world"

tools:context=".MainActivity" />

</RelativeLayout>

This is an example of simple *RelativeLayout* which we will study in a separate chapter. The *TextView* is an Android control used to build the GUI and it have various attributes like  *android:layout\_width*, *android:layout\_height* etc which are being used to set its width and height etc.. The *@string* refers to the strings.xml file located in the res/values folder. Hence, @string/hello\_world refers to the hello string defined in the strings.xml file, which is "Hello World!".

## Running the Application

Let's try to run our **Hello World!** application we just created. Assume we had created our **AVD** while doing environment set-up. To run the app from Android studio, open one of our project's activity files and click Run Eclipse Run Icon icon from the tool bar. Android studio installs the app on our AVD and starts it and if everything is fine with our set-up and application, it will display following Emulator window −



# Android Resources Organizing & Accessing

There are many more items which we use to build a good Android application. Apart from coding for the application, we take care of various other **resources** like static content that our code uses, such as bitmaps, colors, layout definitions, user interface strings, animation instructions, and more. These resources are always maintained separately in various sub-directories under **res/** directory of the project.

## Organize resource in Android Studio

MyProject/

app/

manifest/

AndroidManifest.xml

java/

MyActivity.java

**res/**

drawable/

icon.png

layout/

activity\_main.xml

info.xml

values/

strings.xml

|  |  |
| --- | --- |
| **Sr.No.** | **Directory & Resource Type** |
| 1 | **anim/**  XML files that define property animations. They are saved in res/anim/ folder and accessed from the **R.anim** class. |
| 2 | **color/**  XML files that define a state list of colors. They are saved in res/color/ and accessed from the **R.color** class. |
| 3 | **drawable/**  Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animation drawable. They are saved in res/drawable/ and accessed from the **R.drawable** class. |
| 4 | **layout/**  XML files that define a user interface layout. They are saved in res/layout/ and accessed from the **R.layout** class. |
| 5 | **menu/**  XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. They are saved in res/menu/ and accessed from the **R.menu** class. |
| 6 | **raw/**  Arbitrary files to save in their raw form. You need to call *Resources.openRawResource()* with the resource ID, which is *R.raw.filename* to open such raw files. |
| 7 | **values/**  XML files that contain simple values, such as strings, integers, and colors. For example, here are some filename conventions for resources you can create in this directory −   * arrays.xml for resource arrays, and accessed from the **R.array** class. * integers.xml for resource integers, and accessed from the **R.integer** class. * bools.xml for resource boolean, and accessed from the **R.bool** class. * colors.xml for color values, and accessed from the **R.color** class. * dimens.xml for dimension values, and accessed from the **R.dimen** class. * strings.xml for string values, and accessed from the **R.string** class. * styles.xml for styles, and accessed from the **R.style** class. |
| 8 | **xml/**  Arbitrary XML files that can be read at runtime by calling *Resources.getXML()*. You can save various configuration files here which will be used at run time. |

## Alternative Resources

Our application should provide alternative resources to support specific device configurations. For example, we should include alternative drawable resources (i.e.images) for different screen resolution and alternative string resources for different languages. At runtime, Android detects the current device configuration and loads the appropriate resources for your application.

To specify configuration-specific alternatives for a set of resources, follow the following steps –

* Create a new directory in res/ named in the form **<resources\_name>-<config\_qualifier>**. Here **resources\_name** will be any of the resources mentioned in the above table, like layout, drawable etc. The **qualifier** will specify an individual configuration for which these resources are to be used. We can check official documentation for a complete list of qualifiers for different type of resources.
* Save the respective alternative resources in this new directory. The resource files must be named exactly the same as the default resource files as shown in the below example, but these files will have content specific to the alternative. For example though image file name will be same but for high resolution screen, its resolution will be high.

Below is an example which specifies images for a default screen and alternative images for high resolution screen.

MyProject/

app/

manifest/

AndroidManifest.xml

java/

MyActivity.java

**res/**

drawable/

icon.png

background.png

**drawable-hdpi/**

icon.png

background.png

layout/

activity\_main.xml

info.xml

values/

strings.xml

Below is another example which specifies layout for a default language and alternative layout for Arabic language.

MyProject/

app/

manifest/

AndroidManifest.xml

java/

MyActivity.java

**res/**

drawable/

icon.png

background.png

**drawable-hdpi/**

icon.png

background.png

layout/

activity\_main.xml

info.xml

**layout-ar/**

main.xml

values/

strings.xml

## Accessing Resources

During our application development we will need to access defined resources either in our code, or in our layout XML files. Following section explains how to access our resources in both the scenarios −

### Accessing Resources in Code

When our Android application is compiled, a **R** class gets generated, which contains resource IDs for all the resources available in your **res/** directory. We can use R class to access that resource using sub-directory and resource name or directly resource ID.

### Example

To access *res/drawable/myimage.png* and set an ImageView we will use following code –

ImageView imageView = (ImageView) findViewById(R.id.myimageview);

imageView.setImageResource(R.drawable.myimage);

Here

First line of the code make use of *R.id.myimageview* to get ImageView defined with id *myimageview* in a Layout file.

Second line of code makes use of*R.drawable.myimage* to get an image with name **myimage** available in drawable sub-directory under **/res**.

### Example

Consider next example where *res/values/strings.xml* has following definition –

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="hello">Hello, World!</string>

</resources>

Now we can set the text on a TextView object with ID msg using a resource ID as follows –

TextView msgTextView = (TextView) findViewById(R.id.msg);

msgTextView.setText(R.string.hello);

### Example

Consider a layout *res/layout/activity\_main.xml* with the following definition –

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:orientation="vertical" >

<TextView android:id="@+id/text"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Hello, I am a TextView" />

<Button android:id="@+id/button"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Hello, I am a Button" />

</LinearLayout>

This application code will load this layout for an Activity, in the onCreate() method as follows –

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main\_activity);

}

### Accessing Resources in XML

Consider the following resource XML *res/values/strings.xml* file that includes a color resource and a string resource –

<?xml version="1.0" encoding="utf-8"?>

<resources>

<color name="opaque\_red">#f00</color>

<string name="hello">Hello!</string>

</resources>

Now we can use these resources in the following layout file to set the text color and text string as follows –

<?xml version="1.0" encoding="utf-8"?>

<EditText xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:textColor=**"@color/opaque\_red"**

android:text=**"@string/hello" />**

**3. Android – Architecture & Components**

**Question**

1. Draw the Architecture of Android with All the Components.
2. Write Down the Definition & Function of:
3. Linux Karnel
4. Android Runtime
5. Java Virtual Machine
6. Delvik Virtual Machine
7. Android Libraries
8. Application Components
9. Write Down the Function of Following Android Libraries:
10. Android.app
11. Android.content
12. Android.database
13. Android.opengl
14. Androd.os
15. Android.view
16. Android.widget
17. Android.webkit.
18. Write Short Notes on the Following Application Framework Key Services:
19. Activity Manager
20. Content Providers
21. Resource Manager
22. Notification Manager
23. View System
24. Write Short Notes on Following Application Components:
25. Activities
26. Services
27. Broadcast Receivers
28. Content Providers
29. Fragments
30. Views
31. Layouts
32. Intents
33. Resources
34. Manifest