## Android Libraries and other Components

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.

## 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.

# 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 –

PublicclassMainActivityextendsActivity{

}

## 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 –

PublicclassMyServiceextendsService{

}

## 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.

PublicclassMyReceiverextendsBroadcastReceiver{

PublicvoidonReceive(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.

PublicclassMyContentProviderextendsContentProvider{

PublicvoidonCreate(){}

}

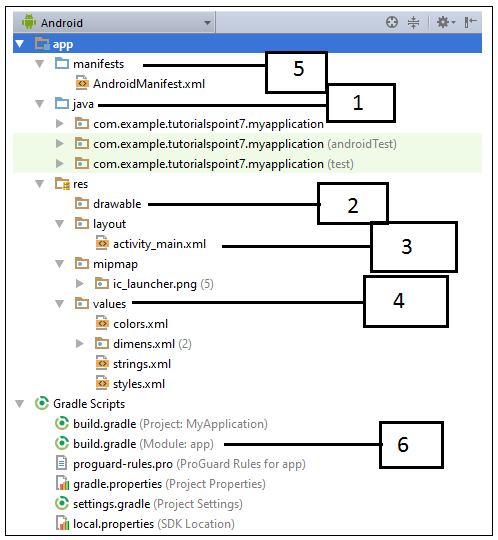
## Additional Components

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. |

**Directories and files in the Android project**

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 |

# 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.

|  |  |
| --- | --- |
| **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. |