ANDROID MANIFEST FILE

The AndroidManifest.xml file contains information of your package, including components of the application such as activities, services, broadcast receivers, content providers etc.

It performs some other tasks also:

- It is responsible to protect the application to access any protected parts by providing the permissions.
- It also declares the android api that the application is going to use.
- It lists the instrumentation classes. The instrumentation classes provide profiling and other information. This information is removed just before the application is published etc.

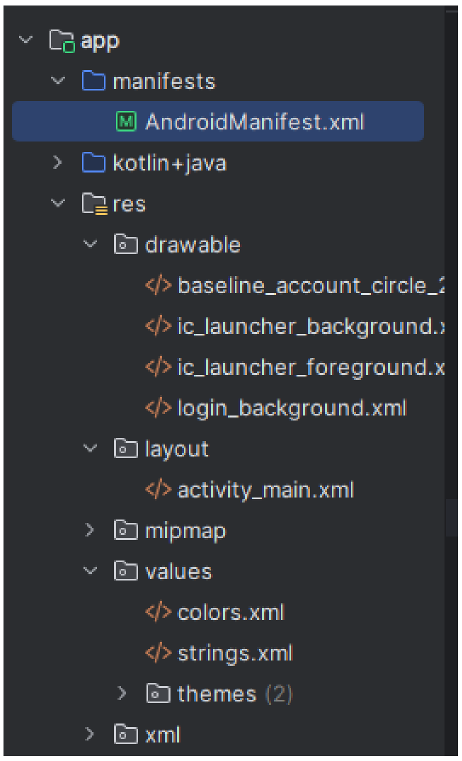
This is the required xml file for all the android application and located inside the root directory:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Maina2"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>

</manifest>
```

EXTERNALIZING RESOURCES

There are many more items which you use to build a good Android application. Apart from coding for the application, you take care of various other resources like static content that your 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



```
app
  manifests
    AndroidManifest.xml
  kotlin+java
  res
    drawable
      baseline_account_circle_
      ic_launcher_background.
      ic_launcher_foreground.x
      login_background.xml
    layout
      activity_main.xml
    mipmap
    values
      colors.xml
      strings.xml
      themes (2)
    xml
```

	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 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. Accessed from the R.menu class.
6	**raw/** Arbitrary files to save in their raw form. You need to call Resources.openRawResource() with the resoufiles
7	**values/** XML files that contain simple values, such as strings, integers, and colors. For example, here are some files 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 configurations.

Alternative Resources

Your application should provide alternative resources to support specific device configurations. For example, you 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 -. 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.
- 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 your application development you will need to access defined resources either in your code, or in your layout XML files. Following section explains how to access your resources in both the scenarios: -

1) Accessing Resources in Code

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

Example 1:

To access res/drawable/myimage.png and set an ImageView you 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 2:

Consider next example where res/values/strings.xml has following definition: -

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

Now you 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 3:

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.activity_main);
}
```

2) 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 you can use these resources in the following layout file to set the text color and text string as follows: -

```xml
<?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" />
```