Mobile Application and Development

Presented By:

Mr. Nitesh Ghodichor

Assistant Processor

Department of Computer Technology, PCE, Nagpur

Using the Android Documentation

Home: This tab provides some high-level news items for Android developers, including announcements of new platform versions. You'll also find quick links for downloading the latest Android SDK, publishing your applications on the Android Market, and other helpful information.

SDK: This tab provides important information about the SDK version installed on your machine. One of the most important features of this tab is the release notes, which describe any known issues for the specific installation. This information is also useful if the online help has been upgraded but you want to develop to an older version of the SDK.

Dev Guide: This tab links to the Android Developer's Guide, which includes a number of FAQs, best practice guides, and a useful glossary of Android terminology for those new to the platform. The appendix section of the Dev Guide tab also details all Android platform versions (API levels), supported media formats, and lists of intents.

Reference: This tab includes a searchable package and class index of all Android APIs provided as part of the Android SDK, in a Javadoc-style format. You will spend most of your time on this tab, looking up Java class documentation, checking method parameters, and other similar tasks.

Resources: This tab includes links to articles, tutorials, and sample code, as well as acts as a gateway to the Android developer forums. There are a number of Google groups you can join, depending on your interests.

Videos: This tab, which is available online only, is your resource for Android training videos. Here, you'll find videos about the Android platform, developer tips, and the Google I/O conference sessions.

Blog: This tab links to the official Android developer blog. Check here for the latest news about the Android platform. This is the place to find how-to examples, learn how to optimize Android applications, and hear about new SDK releases and features as well as Android best practices from the designers of the platform.

Anatomy of Android Application

There are four building blocks to an Android application:

- ***Activity:*** They dictate the UI and handle the user interaction to the smart phone screen.
- ***Intent Receiver***: They handle communication between Android OS and applications.
- ***Service***: They handle background processing associated with an application.
- ***Content Provider***: They handle data and database management issues.

Not every application needs to have all four, but your application will be written with some combination of these.

Once you have decided what components you need for your application, you should list them in a file called AndroidManifest.xml. This is an XML file where you declare the components of your application and what their capabilities and requirements are. We will discuss soon, what the AndroidManifest.xml is responsible for.

Activity :

- Activities are the most common of the four Android building blocks. An activity is usually a single screen in your application. Each activity is implemented as a single class that extends the Activity base class. Your class will display a user interface composed of Views and respond to events. Most applications consist of multiple screens. For example, a text messaging application might have one screen that shows a list of contacts to send messages to, a second screen to write the message to the chosen contact, and other screens to review old messages or change settings. Each of these screens would be implemented as an activity.
- When a new screen opens, the previous screen is paused and put onto a history stack. The user can navigate backward through previously opened screens in the history. Screens can also choose to be removed from the history stack when it would be inappropriate for them to remain. Android retains history stacks for each application launched from the home screen.
- An activity is implemented as a subclass of **Activity** class as follows –

```
public class MainActivity extends Activity { }
```

Intent and Intent Filters :

- Android uses a special class called **Intent** to move from screen to screen. Intent describe what an application wants done. The two most important parts of the intent data structure are the action and the data to act upon. Typical values for action are MAIN (the front door of the application), VIEW, PICK, EDIT, etc. The data is expressed as a Uniform Resource Indicator (URI). For example, to view a website in the browser, you would create an Intent with the VIEW action and the data set to a Website-URI.

```
new Intent(android.content.Intent.VIEW_ACTION,ContentURI.create("http://anddev.org"));
```

- There is a related class called an **IntentFilter**. While an intent is effectively a request to do something, an intent filter is a description of what intents an activity (or intent receiver, see below) is capable of handling. Activities publish their **IntentFilters** in the AndroidManifest.xml file.

Intent Receiver :

- You can use an **IntentReceiver** when you want code in your application to execute in reaction to an external event, for example, when the phone rings, or when the data network is available, or when it's midnight. Intent receivers do not display a UI, although they may display Notifications to alert the user if something interesting has happened. Intent receivers are also registered in AndroidManifest.xml, but you can also register them from code using **Context.registerReceiver().**

Content Provider :

- Applications can store their data in files, a **SQLite database**, preferences or any other mechanism that makes sense. A content provider, however, is useful if you want your application's data to be shared with other applications. A content provider is a class that implements a standard set of methods to let other applications store and retrieve the type of data that is handled by that content provider.

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){}}
```

Service :

- A Service is code that is long-lived and runs without a UI. A good example of this is a media player playing songs from a play list. In a media player application, there would probably be one or more activities that allow the user to choose songs and start playing them. However, the music playback itself should not be handled by an activity because the user will expect the music to keep playing even after navigating to a new screen. In this case, the media player activity could start a service using **Context.startService()** to run in the background to keep the music going. The system will then keep the music playback service running until it has finished. (You can learn more about the priority given to services in the system by reading Life Cycle of an Android Application.) Note that you can connect to a service (and start it if it's not already running) with the **Context.bindService()** method. When connected to a service, you can communicate with it through an interface exposed by the service. For the music service, this might allow you to pause, rewind, etc.

A service is implemented as a subclass of Service class as follows –

```
public class MyService extends Service {}
```

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(){}}
```

AndroidManifest.xml file in android

- 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 provides profiling and other information's. These informations are 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/apkresandroid"

package="com.androidbook.mul
timedia"

android:versionCode="1"

android:versionName="1.0">

<application
android:icon="@drawable/icon"

android:label="@string/app_na
me"

android:debuggable="true">

<activity
android:name=".MultimediaMen
uActivity"

android:label="@string/app_na
me">

<intent-filter>

<action

android:name="android.intent.acti
on.MAIN" />

<category

android:name="android.intent.cat
egory.LAUNCHER" />

</intent-filter>

</activity>

</activity>

<activity
android:name="AudioActivity"></a
ctivity>

<activity
android:name="StillImageActivity"
></activity>

<activity
android:name="VideoPlayActivity"
></activity>

<activity
android:name="VideoRecordActivi
ty"></activity>

</application>

<uses-permission

android:name="android.permissio
n.WRITE_SETTINGS" />

<uses-permission

android:name="android.permissio
n.RECORD_AUDIO" />

<uses-permission

android:name="android.permissio
n.SET_WALLPAPER" />

<uses-permission

android:name="android.permissio
n.CAMERA"></uses-permission>

<uses-sdk

android:minSdkVersion="4"

android:targetSdkVersion="10">

</uses-sdk>

<uses-feature

android:name="android.hardware.
camera" />

</manifest>
```

Elements of the AndroidManifest.xml file, The elements used in the above xml file are described below.

<manifest>

- **manifest** is the root element of the AndroidManifest.xml file. It has **package** attribute that describes the package name of the activity class.

<application>

- **application** is the subelement of the manifest. It includes the namespace declaration. This element contains several subelements that declares the application component such as activity etc.

*The commonly used attributes are of this element are **icon**, **label**, **theme** etc*.

- **android:icon** represents the icon for all the android application components.
- **android:label** works as the default label for all the application components.
- **android:theme** represents a common theme for all the android activities.

<activity>

- **activity** is the subelement of application and represents an activity that must be defined in the AndroidManifest.xml file. It has many attributes such as label, name, theme, launchMode etc.
- **android:label** represents a label i.e. displayed on the screen.
- **android:name** represents a name for the activity class. It is required attribute.

<intent-filter>

- **intent-filter** is the sub-element of activity that describes the type of intent to which activity, service or broadcast receiver can respond to.

<action>

- It adds an action for the intent-filter. The intent-filter must have at least one action element.

<category>

- It adds a category name to an intent-filter.

Android R.java file

- **Android R.java** is *an auto-generated file by aapt* (Android Asset Packaging Tool) that contains resource IDs for all the resources of res/ directory.
- If you create any component in the **activity_main.xml file**, id for the corresponding component is automatically created in this file. This id can be used in the activity source file to perform any action on the component.
- Let's see the **android R.java file**. It includes a lot of static nested classes such as menu, id, layout, attr, drawable, string etc.

Android R.java file

```
package com.example.helloandroid;
public final class R {
    public static final class attr {     }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;     }
    public static final class id {
        public static final int menu_settings=0x7f070000;     }
    public static final class layout {
        public static final int activity_main=0x7f030000;     }
    public static final class menu {
        public static final int activity_main=0x7f060000;     }
    public static final class string {
        public static final int app_name=0x7f040000;
        public static final int hello_world=0x7f040001;
        public static final int menu_settings=0x7f040002;     }

    public static final class style {
  public static final int AppBaseTheme=0x7f050000;
public static final int AppTheme=0x7f050001;     } }
```

Essential Tools for android application development

Essential Tools for android application development

The most important tools required for application development are:

- **JDK**: Java Development kit that is the combination of two premier tools JAVA VIRTUAL MACHINE and JAVA RUNTIME ENVIRONMENT. It helps in compiling and running the java program.
- **ANDROID STUDIO**: The primary goal to introduce the android studio is to develop the android applications in the most stable tool. At first android applications were developed in eclipse but, app developers face multiple issues in eclipse. Then in 2007, google brings the android studio that is used particularly introduced for android.

Terminologies Correlated to Android

XML file

- The preeminent file is used for the structure of an android project. It has complete information about all the components and packages. It initializes the API that is further used by an application.

View

- It is the component of the User Interface that occupies the rectangular area on the screen.

Layout

- It properly aligned the views on the screen.

Activity

- Activity is a User interface screen through which the user interacts. Users have a right to place the UI elements in any way according to the Users choice.

Emulator

- The emulator is the virtual device smartphone provided with an android studio. You can run your created application on the emulator and test its UI and function according to the needs.

Intent

- It acts as a communicating object. You can establish a communication between two or more than two components as services, broadcast receivers. It is used to start and end the activity and services components.

Services

- It is used to run the process even in the background. There is no defined UI for service. Any component can start the service and end the services. You can easily switch between the applications even if the services are running the background.

Content Provider

It implemented in two ways:

- You can use implement the existing content provider in your application.
- However, you can also create a new content provider that will provide or share the data with other applications.
- There are some basic terminologies used in android. Apart from the mentioned terminologies, there are other terms which will frequently use with the android. Once you start learning android, there are a lot of attributes from which you will become aware soon.

Intent Filter in Android

The intent is a messaging object which tells what kind of action to be performed. The intent's most significant use is the launching of the activity. Intent facilitates the communication between the components.

- **Starting Activity :** An activity represents the single screen in an app, Bypassing intent instance we can start an activity.
- **Starting a Service :** A Service is a component that performs operations in the background without a user interface, which is also called a background process.
- **Delivering a Broadcast :** A broadcast is a message that any app can receive. In android, the system delivers various broadcast system events like device starts charging, disable or enable airplane mode, etc.

Intent Type

- There are two types of intent
- **Explicit intent:** Explicit intent can do the specific application action which is set by the code like changing activity, In explicit intent user knows about all the things like after clicking a button which activity will start and Explicit intents are used for communication inside the application
- **Implicit Intent:** Implicit intents do not name a specific component like explicit intent, instead declare general action to perform, which allows a component from another app to handle.

For example: when you tap the share button in any app you can see the Gmail, Bluetooth, and other sharing app options.

Intent Filter

- Implicit intent uses the intent filter to serve the user request.
- The intent filter specifies the types of intents that an activity, service, or broadcast receiver can respond.
- Intent filters are declared in the Android manifest file.
- Intent filter must contain <action>

Most of the intent filter are describe by its

- **<action>, <category>** and **<data>**.

1. <action>

- Adds an action to an intent filter. An <intent-filter> element must contain one or more <action> elements. If there are no <action> elements in an intent filter, the filter doesn't accept any Intent objects.

Syntax : **<action android:name="string" />**

Examples of common action:

- **ACTION_VIEW:** Use this action in intent with startActivity() when you have some information that activity can show to the user like showing an image in a gallery app or an address to view in a map app
- **ACTION_SEND:** You should use this in intent with startActivity() when you have some data that the user can share through another app, such as an email app or social sharing app.

2. <category>

Syntax**: <category android:name="string" />**

- Adds a category name to an intent filter. A string containing additional information about the kind of component that should handle the intent.
- **Example of common categories:**
- **CATEGORY_BROWSABLE:** The target activity allows itself to be started by a web browser to display data referenced by a link.

3. <data>

Syntax:
```
<data android:scheme="string"
      android:host="string"
      android:port="string"
      android:path="string"
      android:pathPattern="string"
      android:pathPrefix="string"
      android:mimeType="string" />
```

Adds a data specification to an intent filter. The specification can be just a data type, just a URI, or both a data type and a URI.

Note: *Uniform Resource Identifier (URI) is a string of characters used to identify a resource. A URI identifies a resource either by location, or a name, or both.*