# MODULE 3

## ANDROID ADVANCED CONCEPTS

### ANDROID: DRAG AND DROP

Android drag/drop framework allows your users to move data from one View to another View in the current layout using a graphical drag and drop gesture. As of **API 11** drag and drop of view onto other views or view groups is supported. The framework includes following three important components to support drag & drop functionality −

- **Drag event class**.
- **Drag listeners**.
- **Helper methods and classes**.

**The Drag/Drop Process**

There are basically four steps or states in the drag and drop process −

- **Started** − This event occurs when you start dragging an item in a layout, your application calls *startDrag()* method to tell the system to start a drag. The arguments inside startDrag() method provide the data to be dragged, metadata for this data, and a callback for drawing the drag shadow.

  The system first responds by calling back to your application to get a drag shadow. It then displays the drag shadow on the device.

  Next, the system sends a drag event with action type *ACTION_DRAG_STARTED* to the registered drag event listeners for all the View objects in the current layout.

  To continue to receive drag events, including a possible drop event, a drag event listener must return **true**, If the drag event listener returns false, then it will not receive drag events for the current operation until the system sends a drag event with action type ACTION_DRAG_ENDED.

- **Continuing** − The user continues the drag. System sends ACTION_DRAG_ENTERED action followed by ACTION_DRAG_LOCATION action to the registered drag event listener for the View where dragging point enters. The listener may choose to alter its View object's appearance in response to the event or can react by highlighting its View.

  The drag event listener receives a ACTION_DRAG_EXITED action after the user has moved the drag shadow outside the bounding box of the View.

- **Dropped** − The user releases the dragged item within the bounding box of a View. The system sends the View object's listener a drag event with action type ACTION_DROP.

- **Ended** − Just after the action type ACTION_DROP, the system sends out a drag event with action type ACTION_DRAG_ENDED to indicate that the drag operation is over.

**The DragEvent Class**

The **DragEvent** represents an event that is sent out by the system at various times during a drag and drop operation. This class provides few Constants and important methods which we use during Drag/Drop process.

**Constants**

Following are all constants integers available as a part of DragEvent class.

| Sr.No. | Constants & Description |
|---|---|
| 1 | **ACTION_DRAG_STARTED**<br>Signals the start of a drag and drop operation. |
| 2 | **ACTION_DRAG_ENTERED**<br>Signals to a View that the drag point has entered the bounding box of the View. |
| 3 | **ACTION_DRAG_LOCATION**<br>Sent to a View after ACTION_DRAG_ENTERED if the drag shadow is still within the View object's bounding box. |
| 4 | **ACTION_DRAG_EXITED**<br>Signals that the user has moved the drag shadow outside the bounding box of the View. |
| 5 | **ACTION_DROP**<br>Signals to a View that the user has released the drag shadow, and the drag point is within the bounding box of the View. |

| | |
|---|---|
| 6 | **ACTION_DRAG_ENDED**<br>Signals to a View that the drag and drop operation has concluded. |

**Methods**

Following are few important and most frequently used methods available as a part of DragEvent class.

| Sr.No. | Constants & Description |
|---|---|
| 1 | **int getAction()**<br>Inspect the action value of this event.. |
| 2 | **ClipData getClipData()**<br>Returns the ClipData object sent to the system as part of the call to startDrag(). |
| 3 | **ClipDescription getClipDescription()**<br>Returns the ClipDescription object contained in the ClipData. |
| 4 | **boolean getResult()**<br>Returns an indication of the result of the drag and drop operation. |
| 5 | **float getX()**<br>Gets the X coordinate of the drag point. |
| 6 | **float getY()**<br>Gets the Y coordinate of the drag point. |
| 7 | **String toString()**<br>Returns a string representation of this DragEvent object. |

**Listening for Drag Event**

If you want any of your views within a Layout should respond Drag event then your view either implements **View.OnDragListener or setup onDragEvent(DragEvent) callback method.** When the system calls the method or listener, it passes to them a DragEvent object explained above. You can have both a listener and a callback method for View object. If this occurs, the system first calls the listener and then defined callback as long as listener returns true.

The combination of the *onDragEvent(DragEvent)* method and *View.OnDragListener* is analogous to the combination of the **onTouchEvent()** and **View.OnTouchListener** used with touch events in old versions of Android.

**Starting a Drag Event**

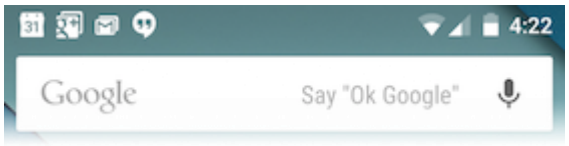
You start with creating a **ClipData** and **ClipData.Item** for the data being moved. As part of the *ClipData* object, supply metadata that is stored in a **ClipDescription** object within the ClipData. For a drag and drop operation that does not represent data movement, you may want to use **null** instead of an actual object.

Next either you can extend extend **View.DragShadowBuilder** to create a drag shadow for dragging the view or simply you can use *View.DragShadowBuilder(View)* to create a default drag shadow that's the same size as the View argument passed to it, with the touch point centered in the drag shadow.

## Android - Notifications

A **notification** is a message you can display to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the notification area. To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.

Android **Toast** class provides a handy way to show users alerts but problem is that these alerts are not persistent which means alert flashes on the screen for a few seconds and then disappears.




To see the details of the notification, you will have to select the icon which will display notification drawer having detail about the notification. While working with emulator with virtual device, you will have to click and drag down the status bar to expand it which will give you detail as follows. This will be just **64 dp** tall and called normal view.


Above expanded form can have a **Big View** which will have additional detail about the notification. You can add upto six additional lines in the notification. The following screen shot shows such notification.

**Create and Send Notifications**

You have simple way to create a notification. Follow the following steps in your application to create a notification −

**Step 1 - Create Notification Builder**

As a first step is to create a notification builder using *NotificationCompat.Builder.build()*. You will use Notification Builder to set various Notification properties like its small and large icons, title, priority etc.

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)
```

**Step 2 - Setting Notification Properties**

Once you have **Builder** object, you can set its Notification properties using Builder object as per your requirement. But this is mandatory to set at least following −

- A small icon, set by **setSmallIcon()**
- A title, set by **setContentTitle()**
- Detail text, set by **setContentText()**

```
mBuilder.setSmallIcon(R.drawable.notification_icon);
mBuilder.setContentTitle("Notification Alert, Click Me!");
mBuilder.setContentText("Hi, This is Android Notification Detail!");
```

You have plenty of optional properties which you can set for your notification.

**Step 3 - Attach Actions**

This is an optional part and required if you want to attach an action with the notification. An action allows users to go directly from the notification to an **Activity** in your application, where they can look at one or more events or do further work.

The action is defined by a **PendingIntent** containing an **Intent** that starts an Activity in your application. To associate the PendingIntent with a gesture, call the appropriate method of *NotificationCompat.Builder*. For example, if you want to start Activity when the user clicks the notification text in the notification drawer, you add the PendingIntent by calling **setContentIntent()**.

A PendingIntent object helps you to perform an action on your applications behalf, often at a later time, without caring of whether or not your application is running.

We take help of stack builder object which will contain an artificial back stack for the started Activity. This ensures that navigating backward from the Activity leads out of your application to the Home screen.

```
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
stackBuilder.addParentStack(ResultActivity.class);

// Adds the Intent that starts the Activity to the top of the stack

stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
stackBuilder.getPendingIntent(0,PendingIntent.FLAG_UPDATE_CURRENT);
mBuilder.setContentIntent(resultPendingIntent);
```

**Step 4 - Issue the notification**

Finally, you pass the Notification object to the system by calling NotificationManager.notify() to send your notification. Make sure you call **NotificationCompat.Builder.build()** method on builder object before notifying it. This method combines all of the options that have been set and return a new **Notification** object.

```
NotificationManager mNotificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);

// notificationID allows you to update the notification later on.
mNotificationManager.notify(notificationID, mBuilder.build());
```

**The NotificationCompat.Builder Class**

The NotificationCompat.Builder class allows easier control over all the flags, as well as help constructing the typical notification layouts. Following are few important and most frequently used methods available as a part of NotificationCompat.Builder class.

| Sl.No. | Constants & Description |
|---|---|
| 1 | **Notification build()**<br>Combine all of the options that have been set and return a new Notification object. |
| 2 | **NotificationCompat.Builder setAutoCancel (boolean autoCancel)**<br>Setting this flag will make it so the notification is automatically canceled when the user clicks it in the panel. |
| 3 | **NotificationCompat.Builder setContent (RemoteViews views)**<br>Supply a custom RemoteViews to use instead of the standard one. |
| 4 | **NotificationCompat.Builder setContentInfo (CharSequence info)**<br>Set the large text at the right-hand side of the notification. |
| 5 | **NotificationCompat.Builder setContentIntent (PendingIntent intent)**<br>Supply a PendingIntent to send when the notification is clicked. |
| 6 | **NotificationCompat.Builder setContentText (CharSequence text)**<br>Set the text (second row) of the notification, in a standard notification. |
| 7 | **NotificationCompat.Builder setContentTitle (CharSequence title)**<br>Set the text (first row) of the notification, in a standard notification. |

| | |
|---|---|
| 8 | **NotificationCompat.Builder setDefaults (int defaults)**<br>Set the default notification options that will be used. |
| 9 | **NotificationCompat.Builder setLargeIcon (Bitmap icon)**<br>Set the large icon that is shown in the ticker and notification. |
| 10 | **NotificationCompat.Builder setNumber (int number)**<br>Set the large number at the right-hand side of the notification. |
| 11 | **NotificationCompat.Builder setOngoing (boolean ongoing)**<br>Set whether this is an ongoing notification. |
| 12 | **NotificationCompat.Builder setSmallIcon (int icon)**<br>Set the small icon to use in the notification layouts. |
| 13 | **NotificationCompat.Builder setStyle (NotificationCompat.Style style)**<br>Add a rich notification style to be applied at build time. |
| 14 | **NotificationCompat.Builder setTicker (CharSequence tickerText)**<br>Set the text that is displayed in the status bar when the notification first arrives. |
| 15 | **NotificationCompat.Builder setVibrate (long[] pattern)**<br>Set the vibration pattern to use. |
| 16 | **NotificationCompat.Builder setWhen (long when)**<br>Set the time that the event occurred. Notifications in the panel are sorted by this time. |

## Android - Location Based Services

Android location APIs make it easy for you to build location-aware applications, without needing to focus on the details of the underlying location technology.

This becomes possible with the help of **Google Play services**, which facilitates adding location awareness to your app with automated location tracking, geofencing, and activity recognition.

### The Location Object

The **Location** object represents a geographic location which can consist of a latitude, longitude, time stamp, and other information such as bearing, altitude and velocity. There are following important methods which you can use with Location object to get location specific information –

| Sl.No. | Method & Description |
|---|---|
| 1 | **float distanceTo(Location dest)**<br>Returns the approximate distance in meters between this location and the given location. |
| 2 | **float getAccuracy()**<br>Get the estimated accuracy of this location, in meters. |
| 3 | **double getAltitude()**<br>Get the altitude if available, in meters above sea level. |
| 4 | **float getBearing()**<br>Get the bearing, in degrees. |
| 5 | **double getLatitude()**<br>Get the latitude, in degrees. |
| 6 | **double getLongitude()** |

| | |
|---|---|
| | Get the longitude, in degrees. |
| 7 | **float getSpeed()**<br>Get the speed if it is available, in meters/second over ground. |
| 8 | **boolean hasAccuracy()**<br>True if this location has an accuracy. |
| 9 | **boolean hasAltitude()**<br>True if this location has an altitude. |
| 10 | **boolean hasBearing()**<br>True if this location has a bearing. |
| 11 | **boolean hasSpeed()**<br>True if this location has a speed. |
| 12 | **void reset()**<br>Clears the contents of the location. |
| 13 | **void setAccuracy(float accuracy)**<br>Set the estimated accuracy of this location, meters. |
| 14 | **void setAltitude(double altitude)**<br>Set the altitude, in meters above sea level. |
| 15 | **void setBearing(float bearing)**<br>Set the bearing, in degrees. |

| | |
|---|---|
| 16 | **void setLatitude(double latitude)**<br>Set the latitude, in degrees. |
| 17 | **void setLongitude(double longitude)**<br>Set the longitude, in degrees. |
| 18 | **void setSpeed(float speed)**<br>Set the speed, in meters/second over ground. |
| 19 | **String toString()**<br>Returns a string containing a concise, human-readable description of this object. |

**Get the Current Location**

To get the current location, create a location client which is **LocationClient** object, connect it to Location Services using **connect()** method, and then call its **getLastLocation()** method. This method returns the most recent location in the form of **Location** object that contains latitude and longitude coordinates and other information as explained above. To have location based functionality in your activity, you will have to implement two interfaces −

- GooglePlayServicesClient.ConnectionCallbacks
- GooglePlayServicesClient.OnConnectionFailedListener

These interfaces provide following important callback methods, which you need to implement in your activity class –

| Sr.No. | Callback Methods & Description |
|---|---|
| 1 | **abstract void onConnected(Bundle connectionHint)**<br>This callback method is called when location service is connected to the location client |

| | |
|---|---|
| | successfully. You will use **connect()** method to connect to the location client. |
| 2 | **abstract void onDisconnected()**<br>This callback method is called when the client is disconnected. You will use **disconnect()** method to disconnect from the location client. |
| 3 | **abstract void onConnectionFailed(ConnectionResult result)**<br>This callback method is called when there was an error connecting the client to the service. |

You should create the location client in **onCreate()** method of your activity class, then connect it in **onStart()**, so that Location Services maintains the current location while your activity is fully visible. You should disconnect the client in **onStop()** method, so that when your app is not visible, Location Services is not maintaining the current location. This helps in saving battery power up-to a large extent.

**Get the Updated Location**

If you are willing to have location updates, then apart from above mentioned interfaces, you will need to implement **LocationListener** interface as well. This interface provide following callback method, which you need to implement in your activity class −

| Sr.No. | Callback Method & Description |
|---|---|
| 1 | **abstract void onLocationChanged(Location location)**<br>This callback method is used for receiving notifications from the LocationClient when the location has changed. |

**Location Quality of Service**

The **LocationRequest** object is used to request a quality of service (QoS) for location updates from the **LocationClient**. There are following useful setter methods which you can use to handle QoS. There are equivalent getter methods available which you can check in Android official documentation.

| Sl.No. | Method & Description |
|---|---|
| 1 | **setExpirationDuration(long millis)**<br>Set the duration of this request, in milliseconds. |
| 2 | **setExpirationTime(long millis)**<br>Set the request expiration time, in millisecond since boot. |
| 3 | **setFastestInterval(long millis)**<br>Explicitly set the fastest interval for location updates, in milliseconds. |
| 4 | **setInterval(long millis)**<br>Set the desired interval for active location updates, in milliseconds. |
| 5 | **setNumUpdates(int numUpdates)**<br>Set the number of location updates. |
| 6 | **setPriority(int priority)**<br>Set the priority of the request. |

Now for example, if your application wants high accuracy location it should create a location request with **setPriority(int)** set to PRIORITY_HIGH_ACCURACY and **setInterval(long)** to 5 seconds. You can also use bigger interval and/or other priorities like PRIORITY_LOW_POWER for to request "city" level accuracy or PRIORITY_BALANCED_POWER_ACCURACY for "block" level accuracy.

Activities should strongly consider removing all location request when entering the background (for example at onPause()), or at least swap the request to a larger interval and lower quality to save power consumption.

**Displaying a Location Address**

Once you have **Location** object, you can use **Geocoder.getFromLocation()** method to get an address for a given latitude and longitude. This method is synchronous, and may take a long time to do its work, so you should call the method from the **doInBackground()** method of an **AsyncTask** class.

The **AsyncTask** must be subclassed to be used and the subclass will override **doInBackground(Params...)** method to perform a task in the background and **onPostExecute(Result)** method is invoked on the UI thread after the background computation finishes and at the time to display the result. There is one more important method available in AyncTask which is **execute(Params... params)**, this method executes the task with the specified parameters.

## Android - Sending Email

**Email** is messages distributed by electronic means from one system user to one or more recipients via a network.

Before starting Email Activity, You must know Email functionality with intent, Intent is carrying data from one component to another component with-in the application or outside the application.

To send an email from your application, you don't have to implement an email client from the beginning, but you can use an existing one like the default Email app provided from Android, Gmail, Outlook, K-9 Mail etc. For this purpose, we need to write an Activity that launches an email client, using an implicit Intent with the right action and data. In this example, we are going to send an email from our app by using an Intent object that launches existing email clients.

Following section explains different parts of our Intent object required to send an email.

**Intent Object - Action to send Email**

You will use **ACTION_SEND** action to launch an email client installed on your Android device. Following is simple syntax to create an intent with ACTION_SEND action.

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);
```

**Intent Object - Data/Type to send Email**

To send an email you need to specify **mailto:** as URI using setData() method and data type will be to **text/plain** using setType() method as follows −

```
emailIntent.setData(Uri.parse("mailto:"));
emailIntent.setType("text/plain");
```

**Intent Object - Extra to send Email**

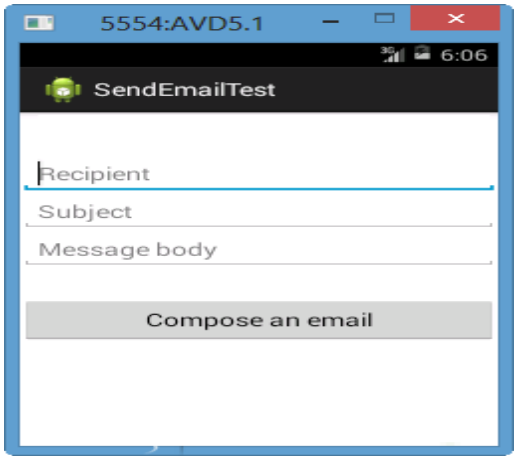
Android has built-in support to add TO, SUBJECT, CC, TEXT etc. fields which can be attached to the intent before sending the intent to a target email client. You can use following extra fields in your email –

| Sl.No. | Extra Data & Description |
|---|---|
| 1 | **EXTRA_BCC**<br>A String[] holding e-mail addresses that should be blind carbon copied. |
| 2 | **EXTRA_CC**<br>A String[] holding e-mail addresses that should be carbon copied. |
| 3 | **EXTRA_EMAIL**<br>A String[] holding e-mail addresses that should be delivered to. |
| 4 | **EXTRA_HTML_TEXT**<br>A constant String that is associated with the Intent, used with ACTION_SEND to supply an alternative to EXTRA_TEXT as HTML formatted text. |
| 5 | **EXTRA_SUBJECT**<br>A constant string holding the desired subject line of a message. |
| 6 | **EXTRA_TEXT**<br>A constant CharSequence that is associated with the Intent, used with ACTION_SEND to supply the literal data to be sent. |
| 7 | **EXTRA_TITLE**<br>A CharSequence dialog title to provide to the user when used with a ACTION_CHOOSER. |

Here is an example showing you how to assign extra data to your intent −

```
emailIntent.putExtra(Intent.EXTRA_EMAIL  , new String[]{"Recipient"});
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "subject");
emailIntent.putExtra(Intent.EXTRA_TEXT   , "Message Body");
```

The out-put of above code is as below shown an image




## Android - Sending SMS

In Android, we can use SmsManager API or devices Built-in SMS application to send SMS's. The two basic examples to send SMS message −

**SmsManager API**

```
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage("phoneNo", null, "sms message", null, null);
```

**Built-in SMS application**

```
Intent sendIntent = new Intent(Intent.ACTION_VIEW);
sendIntent.putExtra("sms_body", "default content");
sendIntent.setType("vnd.android-dir/mms-sms");
startActivity(sendIntent);
```

REMEMBER both need **SEND_SMS permission**.

<uses-permission android:name="android.permission.SEND_SMS" />

Apart from the above method, there are few other important functions available in SmsManager class. These methods are listed below –

| Sr.No. | Method & Description |
|---|---|
| 1 | **ArrayList<String> divideMessage(String text)**<br>This method divides a message text into several fragments, none bigger than the maximum SMS message size. |
| 2 | **static SmsManager getDefault()**<br>This method is used to get the default instance of the SmsManager |
| 3 | **void sendDataMessage(String destinationAddress, String scAddress, short destinationPort, byte[] data, PendingIntent sentIntent, PendingIntent deliveryIntent)**<br>This method is used to send a data based SMS to a specific application port. |
| 4 | **void sendMultipartTextMessage(String destinationAddress, String scAddress, ArrayList<String> parts, ArrayList<PendingIntent> sentIntents, ArrayList<PendingIntent> deliveryIntents)**<br>Send a multi-part text based SMS. |
| 5 | **void sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)**<br>Send a text based SMS. |

## Android - Phone Calls

Android provides Built-in applications for phone calls, in some occasions we may need to make a phone call through our application. This could easily be done by using implicit Intent with

appropriate actions. Also, we can use PhoneStateListener and TelephonyManager classes, in order to monitor the changes in some telephony states on the device.

We can use Android Intent to make phone call by calling built-in Phone Call functionality of the Android. Following section explains different parts of our Intent object required to make a call.

**Intent Object - Action to make Phone Call**

You will use **ACTION_CALL** action to trigger built-in phone call functionality available in Android device. Following is simple syntax to create an intent with ACTION_CALL action

```
Intent phoneIntent = new Intent(Intent.ACTION_CALL);
```

You can use **ACTION_DIAL** action instead of ACTION_CALL, in that case you will have option to modify hardcoded phone number before making a call instead of making a direct call.

**Intent Object - Data/Type to make Phone Call**

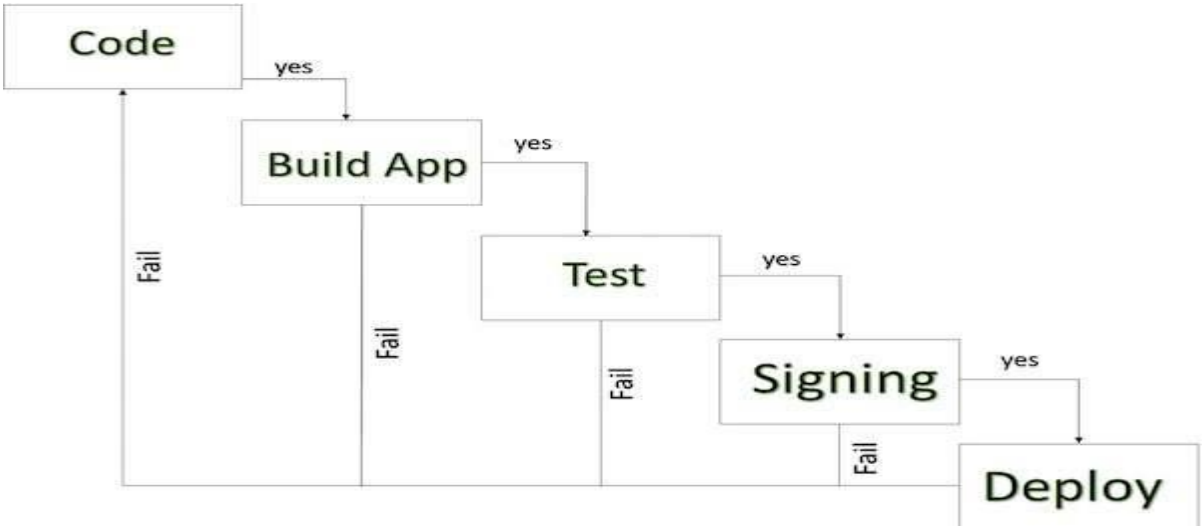
To make a phone call at a given number 91-000-000-0000, you need to specify **tel:** as URI using setData() method as follows −

```
phoneIntent.setData(Uri.parse("tel:91-000-000-0000"));
```

The interesting point is that, to make a phone call, you do not need to specify any extra data or data type.

## Publishing Android Application

Android application publishing is a process that makes your Android applications available to users. Infect, publishing is the last phase of the Android application development process.




**Android development life cycle**

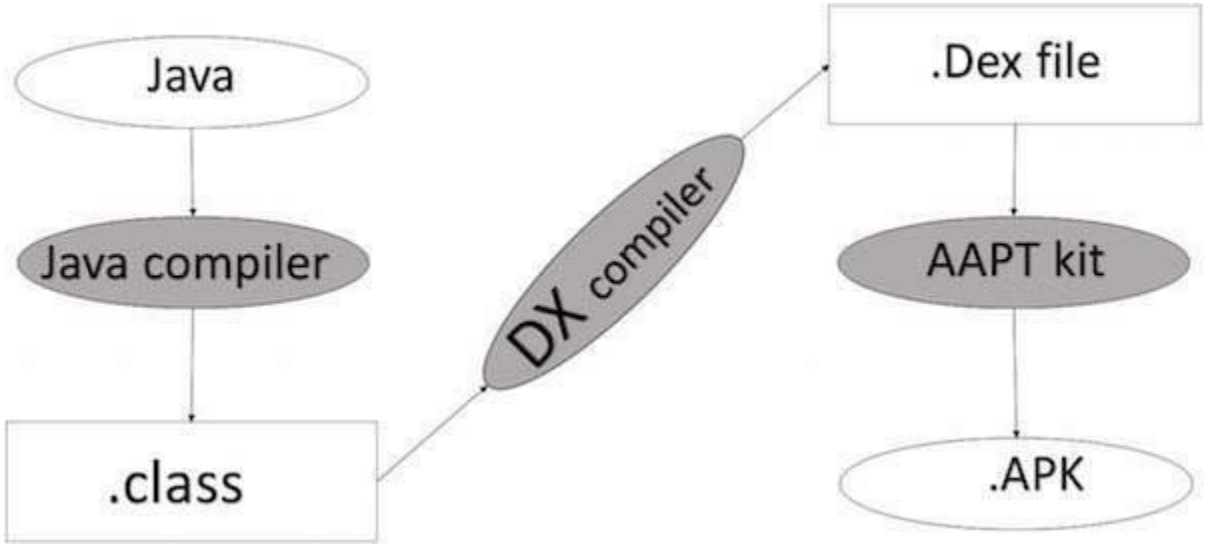
Once you developed and fully tested your Android Application, you can start selling or distributing free using Google Play (A famous Android marketplace). We can also release your applications by sending them directly to users or by letting users download them from your own website.

Here is a simplified check list which will help you in launching your Android application −

| Step | Activity |
|---|---|
| 1 | **Regression Testing** Before you publish your application, you need to make sure that its meeting the basic quality expectations for all Android apps, on all of the devices that you are targeting. So perform all the required testing on different devices including phone and tablets. |
| 2 | **Application Rating** When you will publish your application at Google Play, you will have to specify a content rating for your app, which informs Google Play users of its maturity level. Currently available ratings are (a) Everyone (b) Low maturity (c) Medium maturity (d) High maturity. |
| 3 | **Targeted Regions** Google Play lets you control what countries and territories where your application will be sold. Accordingly you must take care of setting up time zone, localization or any other specific requirement as per the targeted region. |

| | |
|---|---|
| 4 | **Application Size** Currently, the maximum size for an APK published on Google Play is 50 MB. If your app exceeds that size, or if you want to offer a secondary download, you can use APK Expansion Files, which Google Play will host for free on its server infrastructure and automatically handle the download to devices. |
| 5 | **SDK and Screen Compatibility** It is important to make sure that your app is designed to run properly on the Android platform versions and device screen sizes that you want to target. |
| 6 | **Application Pricing** Deciding whether you app will be free or paid is important because, on Google Play, free app's must remain free. If you want to sell your application then you will have to specify its price in different currencies. |
| 7 | **Promotional Content** It is a good marketing practice to supply a variety of high-quality graphic assets to showcase your app or brand. After you publish, these appear on your product details page, in store listings and search results, and elsewhere. |
| 8 | **Build and Upload release-ready APK** The release-ready APK is what you you will upload to the Developer Console and distribute to users. You can check complete detail on how to create a release-ready version of your app: Preparing for Release. |
| 9 | **Finalize Application Detail** Google Play gives you a variety of ways to promote your app and engage with users on your product details page, from colourful graphics, screen shots, and videos to localized descriptions, release details, and links to your other apps. So you can decorate your application page and provide as much as clear crisp detail you can provide. |

**Export Android Application Process**



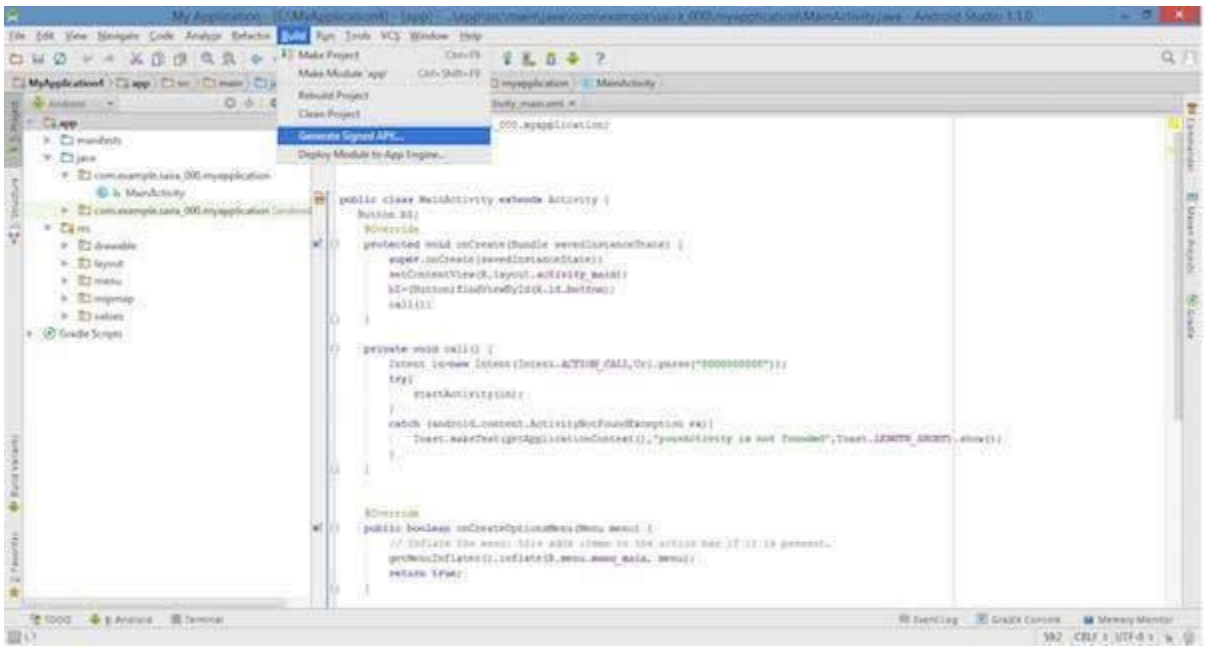

***Apk development process***

Before exporting the apps, you must some of tools

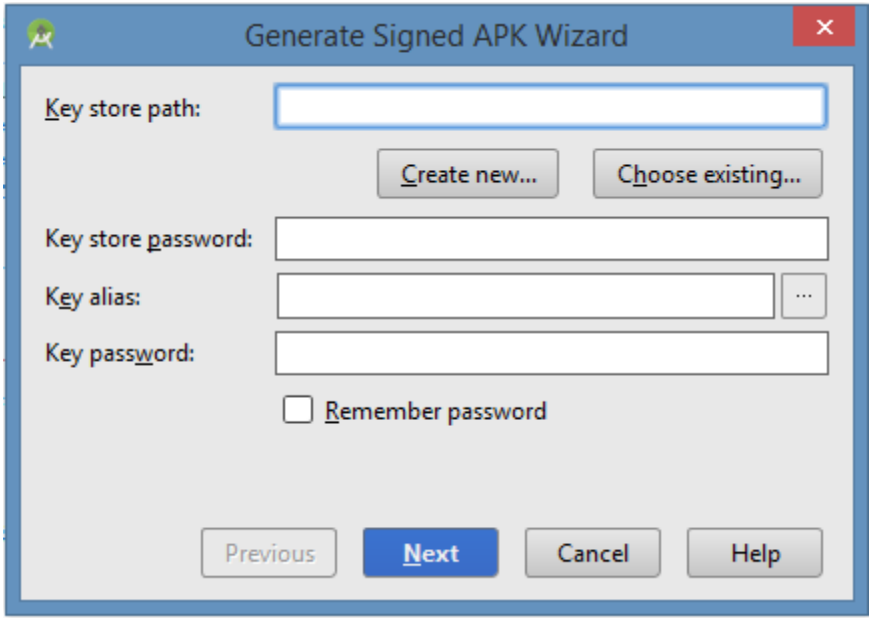
- **Dx tools**(Dalvik executable tools ): It going to convert **.class file** to **.dex file**. it has useful for memory optimization and reduce the boot-up speed time
- **AAPT**(Android assistance packaging tool):it has useful to convert **.Dex file** to**.Apk**
- **APK**(Android packaging kit): The final stage of deployment process is called as .apk.

We have to export your application as an APK (Android Package) file before you upload it Google Play marketplace.

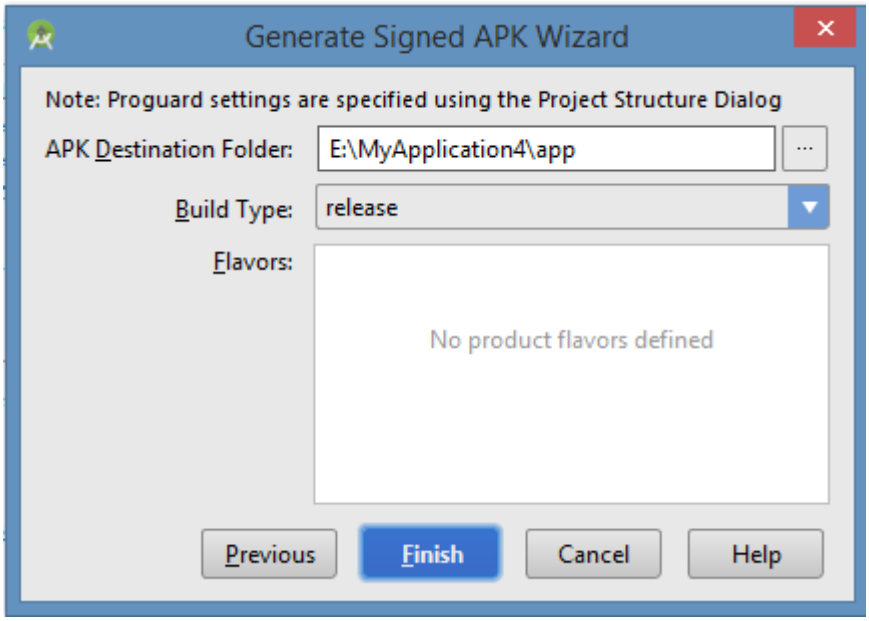
To export an application, just open that application project in Android studio and select **Build → Generate Signed APK** from your Android studio and follow the simple steps to export your application −



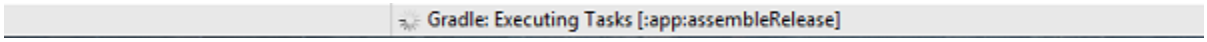
Next select, **Generate Signed APK** option as shown in the above screen shot and then click it so that you get following screen where you will choose **Create new keystore** to store your application.




Enter your key store path,key store password,key alias and key password to protect your application and click on **Next** button once again. It will display following screen to let you create an application –




Once you filled up all the information,like app destination,build type and flavours click **finish** button While creating an application it will show as below




Finally, it will generate your Android Application as APK formate File which will be uploaded at Google Play marketplace.

Google Play Registration

The most important step is to register with Google Play using Google Play Marketplace. You can use your existing google ID if you have any otherwise you can create a new Google ID and then register with the marketplace. You will have following screen to accept terms and condition.


You can use **Continue to payment** button to proceed to make a payment of $25 as a registration fee and finally to complete your account detail.

Once you are a registered user at Google Play, you can upload **release-ready APK** for your application and finally you will complete application detail using application detail page as mentioned in step 9 of the above mentioned checklist.

Signing Your App Manually

You do not need Android Studio to sign your app. You can sign your app from the command line using standard tools from the Android SDK and the JDK. To sign an app in release mode from the command line −

- **Generate a private key using keytool**

```
$ keytool -genkey -v -keystore my-release-key.keystore
-alias alias_name -keyalg RSA -keysize 2048 -validity 10000
```

- **Compile your app in release mode to obtain an unsigned APK**
- **Sign your app with your private key using jarsigner**

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1
-keystore my-release-key.keystore my_application.apk alias_name
```

- **Verify that your APK is signed. For example –**

```
$ jarsigner -verify -verbose -certs my_application.apk
```

- **Align the final APK package using zipalign.**

```
$ zipalign -v 4 your_project_name-unaligned.apk your_project_name.apk
```

## Android - Alert Dialog

A Dialog is small window that prompts the user to a decision or enter additional information.

Some times in your application, if you wanted to ask the user about taking a decision between yes or no in response of any particular action taken by the user, by remaining in the same activity and without changing the screen, you can use Alert Dialog.

In order to make an alert dialog, you need to make an object of AlertDialogBuilder which an inner class of AlertDialog. Its syntax is given below

```
AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this);
```

Now you have to set the positive (yes) or negative (no) button using the object of the AlertDialogBuilder class. Its syntax is

```
alertDialogBuilder.setPositiveButton(CharSequence text,
  DialogInterface.OnClickListener listener)
alertDialogBuilder.setNegativeButton(CharSequence text,
  DialogInterface.OnClickListener listener)
```

Apart from this , we can use other functions provided by the builder class to customize the alert dialog. These are listed below

| Sr.No | Method type & description |
|---|---|
| 1 | **setIcon(Drawable icon)**<br>This method set the icon of the alert dialog box. |
| 2 | **setCancelable(boolean cancel able)**<br>This method sets the property that the dialog can be cancelled or not |
| 3 | **setMessage(CharSequence message)**<br>This method sets the message to be displayed in the alert dialog |
| 4 | **setMultiChoiceItems(CharSequence[] items, boolean[] checkedItems, DialogInterface.OnMultiChoiceClickListener listener)**<br>This method sets list of items to be displayed in the dialog as the content. The selected option will be notified by the listener |
| 5 | **setOnCancelListener(DialogInterface.OnCancelListener onCancelListener)**<br>This method Sets the callback that will be called if the dialog is cancelled. |
| 6 | **setTitle(CharSequence title)**<br>This method set the title to be appear in the dialog |

After creating and setting the dialog builder , you will create an alert dialog by calling the create() method of the builder class. Its syntax is

```
AlertDialog alertDialog = alertDialogBuilder.create();
alertDialog.show();
```

This will create the alert dialog and will show it on the screen.

**Dialog fragment**

Dialog fragment is a fragment which can show fragment in dialog box

**List dialog**

It has used to show list of items in a dialog box.For suppose, user need to select a list of items or else need to click a item from multiple list of items.At this situation we can use list dialog.

**Single-choice list dialog**

It has used to add single choice list to Dialog box.We can check or uncheck as per user choice.

## Android - Animations

Animation is the process of creating motion and shape change

Animation in android is possible from many ways. In this chapter we will discuss one easy and widely used way of making animation called tweened animation.

**Tween Animation**

Tween Animation takes some parameters such as start value , end value, size , time duration , rotation angle e.t.c and perform the required animation on that object. It can be applied to any type of object. So in order to use this , android has provided us a class called Animation.

In order to perform animation in android , we are going to call a static function loadAnimation() of the class AnimationUtils. We are going to receive the result in an instance of Animation Object. Its syntax is as follows −

```
Animation animation = AnimationUtils.loadAnimation(getApplicationContext(),
  R.anim.myanimation);
```

Note the second parameter. It is the name of the our animation xml file. You have to create a new folder called **anim** under res directory and make an xml file under anim folder.

This animation class has many useful functions which are listed below −

| Sr.No | Method & Description |
|---|---|
| 1 | **start()**<br>This method starts the animation. |
| 2 | **setDuration(long duration)**<br>This method sets the duration of an animation. |
| 3 | **getDuration()**<br>This method gets the duration which is set by above method |
| 4 | **end()**<br>This method ends the animation. |
| 5 | **cancel()**<br>This method cancels the animation. |

In order to apply this animation to an object , we will just call the startAnimation() method of the object. Its syntax is −

```
ImageView image1 = (ImageView)findViewById(R.id.imageView1);
image.startAnimation(animation);
```

## Android - Audio Capture

Android has a built in microphone through which you can capture audio and store it , or play it in your phone. There are many ways to do that but the most common way is through MediaRecorder class.

Android provides MediaRecorder class to record audio or video. In order to use MediaRecorder class ,you will first create an instance of MediaRecorder class. Its syntax is given below.

```
MediaRecorder myAudioRecorder = new MediaRecorder();
```

Now set the source , output and encoding format and output file. Their syntax is given below.

```
myAudioRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
myAudioRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
myAudioRecorder.setAudioEncoder(MediaRecorder.OutputFormat.AMR_NB);
myAudioRecorder.setOutputFile(outputFile);
```

After specifying the audio source and format and its output file, we can then call the two basic methods prepare and start to start recording the audio.

```
myAudioRecorder.prepare();
myAudioRecorder.start();
```

Apart from these methods , there are other methods listed in the MediaRecorder class that allows you more control over audio and video recording.

| Sr.No | Method & description |
|---|---|
| 1 | **setAudioSource()**<br>This method specifies the source of audio to be recorded |
| 2 | **setVideoSource()**<br>This method specifies the source of video to be recorded |
| 3 | **setOutputFormat()**<br>This method specifies the audio format in which audio to be stored |
| 4 | **setAudioEncoder()**<br>This method specifies the audio encoder to be used |
| 5 | **setOutputFile()**<br>This method configures the path to the file into which the recorded audio is to be stored |
| 6 | **stop()** |

| | |
|---|---|
| | This method stops the recording process. |
| 7 | **release()**<br>This method should be called when the recorder instance is needed. |

## Android - Audio Manager

We can easily control your ringer volume and ringer profile i-e:(silent,vibrate,loud e.t.c) in android. Android provides AudioManager class that provides access to these controls.

In order to use AndroidManager class, you have to first create an object of AudioManager class by calling the **getSystemService()** method. Its syntax is given below.

```
private AudioManager myAudioManager;
myAudioManager = (AudioManager)getSystemService(Context.AUDIO_SERVICE);
```

Once you instantiate the object of AudioManager class, you can use **setRingerMode** method to set the audio or ringer profile of your device. Its syntax is given below.

```
myAudioManager.setRingerMode(AudioManager.RINGER_MODE_VIBRATE);
```

The method setRingerMode takes an integer number as a parameter. For each mode , an integer number is assigned that will differentiate between different modes. The possible modes are.

| Sr.No | Mode & Description |
|---|---|
| 1 | **RINGER_MODE_VIBRATE**<br>This Mode sets the device at vibrate mode. |
| 2 | **RINGER_MODE_NORMAL**<br>This Mode sets the device at normal(loud) mode. |
| 3 | **RINGER_MODE_SILENT**<br>This Mode sets the device at silent mode. |

Once you have set the mode , you can call the **getRingerMode()** method to get the set state of the system. Its syntax is given below.

```
int mod = myAudioManager.getRingerMode();
```

Apart from the getRingerMode method, there are other methods available in the AudioManager class to control the volume and other modes. They are listed below.

| Sr.No | Method & description |
|---|---|
| 1 | **adjustVolume(int direction, int flags)**<br>This method adjusts the volume of the most relevant stream |
| 2 | **getMode()**<br>This method returns the current audio mode |
| 3 | **getStreamMaxVolume(int streamType)**<br>This method returns the maximum volume index for a particular stream |
| 4 | **getStreamVolume(int streamType)**<br>This method returns the current volume index for a particular stream |
| 5 | **isMusicActive()**<br>This method checks whether any music is active. |
| 6 | **startBluetoothSco()**<br>This method Start bluetooth SCO audio connection |
| 7 | **stopBluetoothSco()**<br>This method stop bluetooth SCO audio connection. |

## Android - Auto Complete

If you want to get suggestions , when you type in an editable text field , you can do this via AutoCompleteTextView. It provides suggestions automatically when the user is typing. The list of suggestions is displayed in a drop down menu from which the user can choose an item to replace the content of the edit box with.

In order to use AutoCompleteTextView you have to first create an AutoCompletTextView Field in the xml. Its syntax is given below.

```
<AutoCompleteTextView
  android:id="@+id/autoCompleteTextView1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignParentTop="true"
  android:layout_centerHorizontal="true"
  android:layout_marginTop="65dp"
  android:ems="10" >
```

After that, you have to get a reference of this textview in java. Its syntax is given below.

```
private AutoCompleteTextView actv;
actv = (AutoCompleteTextView) findViewById(R.id.autoCompleteTextView1);
```

The the next thing you need to do is to specify the list of suggestions items to be displayed. You can specify the list items as a string array in java or in strings.xml. Its syntax is given below.

```
String[] countries = getResources().getStringArray(R.array.list_of_countries);
ArrayAdapter<String> adapter = new ArrayAdapter<String>
  (this,android.R.layout.simple_list_item_1,countries);
actv.setAdapter(adapter);
```

The array adapter class is responsible for displaying the data as list in the suggestion box of the text field. The **setAdapter** method is used to set the adapter of the autoCompleteTextView. Apart from these methods, the other methods of Auto Complete are listed below.

| Sr.No | Method & description |
|---|---|
| 1 | **getAdapter()**<br>This method returns a filterable list adapter used for auto completion |

| | |
|---|---|
| 2 | **getCompletionHint()**<br>This method returns optional hint text displayed at the bottom of the the matching list |
| 3 | **getDropDownAnchor()**<br>This method returns returns the id for the view that the auto-complete drop down list is anchored to. |
| 4 | **getListSelection()**<br>This method returns the position of the dropdown view selection, if there is one |
| 5 | **isPopupShowing()**<br>This method indicates whether the popup menu is showing |
| 6 | **setText(CharSequence text, boolean filter)**<br>This method sets text except that it can disable filtering |
| 7 | **showDropDown()**<br>This method displays the drop down on screen. |

## Android - Best Practices

There are some practices that you can follow while developing android application. These are suggested by the android itself and they keep on improving with respect to time.

These best practices include interaction design features, performance, security and privacy, compatibility, testing, distributing and monetizing tips. They are narrowed down and are listed as below.

**Best Practices - User input**

Every text field is intended for a different job. For example, some text fields are for text and some are for numbers. If it is for numbers then it is better to display the numeric keypad when that textfield is focused. Its syntax is.

```
<EditText
```

```
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/editText"
  android:layout_alignParentRight="true"
  android:layout_alignParentEnd="true"
  android:hint="User Name"
  android:layout_below="@+id/imageView"
  android:layout_alignLeft="@+id/imageView"
  android:layout_alignStart="@+id/imageView"
  android:numeric="integer" />
```

Other then that if your field is for password, then it must show a password hint, so that the user can easily remember the password. It can be achieved as.

```
<EditText
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/editText2"
  android:layout_alignLeft="@+id/editText"
  android:layout_alignStart="@+id/editText"
  android:hint="Pass Word"
  android:layout_below="@+id/editText"
  android:layout_alignRight="@+id/editText"
  android:layout_alignEnd="@+id/editText"
  android:password="true" />
```

**Best Practices - Background jobs**

There are certain jobs in an application that are running in an application background. Their job might be to fetch some thing from the internet , playing music e.t.c. It is recommended that the long awaiting tasks should not be done in the UI thread and rather in the background by services or AsyncTask.

**AsyncTask Vs Services**.

Both are used for doing background tasks , but the service is not affected by most user interface life cycle events, so it continues to run in circumstances that would shut down an AsyncTask.

**Best Practices - Performance**

Your application performance should be up-to the mark. But it should perform differently not on the front end , but on the back end when it the device is connected to a power source or charging. Charging could be of from USB and from wire cable.

When your device is charging itself , it is recommended to update your application settings if any, such as maximizing your refresh rate whenever the device is connected. It can be done as this.

```
IntentFilter ifilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
Intent batteryStatus = context.registerReceiver(null, ifilter);

// Are we charging / charged? Full or charging.
int status = batteryStatus.getIntExtra(BatteryManager.EXTRA_STATUS, -1);

// How are we charging? From AC or USB.
int chargePlug = batteryStatus.getIntExtra(BatteryManager.EXTRA_PLUGGED, -1);
```

**Best Practices - Security and privacy**

It is very important that your application should be secure and not only the application , but the user data and the application data should also be secured. The security can be increased by the following factors.

- Use internal storage rather then external for storing applications files
- Use content providers wherever possible
- Use SSl when connecting to the web
- Use appropriate permissions for accessing different functionalities of device

## Android - Bluetooth

Among many ways, Bluetooth is a way to send or receive data between two different devices. Android platform includes support for the Bluetooth framework that allows a device to wirelessly exchange data with other Bluetooth devices.

Android provides Bluetooth API to perform these different operations.

- Scan for other Bluetooth devices
- Get a list of paired devices
- Connect to other devices through service discovery

Android provides BluetoothAdapter class to communicate with Bluetooth. Create an object of this calling by calling the static method getDefaultAdapter(). Its syntax is given below.

```
private BluetoothAdapter BA;
BA = BluetoothAdapter.getDefaultAdapter();
```

In order to enable the Bluetooth of your device, call the intent with the following Bluetooth constant ACTION_REQUEST_ENABLE. Its syntax is.

```
Intent turnOn = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
```

```
startActivityForResult(turnOn, 0);
```

Apart from this constant, there are other constants provided the API , that supports different tasks. They are listed below.

| Sr.No | Constant & description |
|---|---|
| 1 | **ACTION_REQUEST_DISCOVERABLE**<br>This constant is used for turn on discovering of bluetooth |
| 2 | **ACTION_STATE_CHANGED**<br>This constant will notify that Bluetooth state has been changed |
| 3 | **ACTION_FOUND**<br>This constant is used for receiving information about each device that is discovered |

Once you enable the Bluetooth , you can get a list of paired devices by calling getBondedDevices() method. It returns a set of bluetooth devices. Its syntax is.

```
private Set<BluetoothDevice>pairedDevices;
pairedDevices = BA.getBondedDevices();
```

Apart form the parried Devices , there are other methods in the API that gives more control over Blueetooth. They are listed below.

| Sr.No | Method & description |
|---|---|
| 1 | **enable()**<br>This method enables the adapter if not enabled |
| 2 | **isEnabled()** |

| | |
|---|---|
| | This method returns true if adapter is enabled |
| 3 | **disable()**<br>This method disables the adapter |
| 4 | **getName()**<br>This method returns the name of the Bluetooth adapter |
| 5 | **setName(String name)**<br>This method changes the Bluetooth name |
| 6 | **getState()**<br>This method returns the current state of the Bluetooth Adapter. |
| 7 | **startDiscovery()**<br>This method starts the discovery process of the Bluetooth for 120 seconds. |

## Android - Camera

These are the following two ways, in which you can use camera in your application

- Using existing android camera application in our application
- Directly using Camera API provided by android in our application

### Using existing android camera application in our application

You will use MediaStore.ACTION_IMAGE_CAPTURE to launch an existing camera application installed on your phone. Its syntax is given below

```
Intent intent = new Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
```

Apart from the above, there are other available Intents provided by MediaStore. They are listed as follows

| Sr.No | Intent type and description |
|---|---|
| 1 | **ACTION_IMAGE_CAPTURE_SECURE**<br>It returns the image captured from the camera , when the device is secured |
| 2 | **ACTION_VIDEO_CAPTURE**<br>It calls the existing video application in android to capture video |
| 3 | **EXTRA_SCREEN_ORIENTATION**<br>It is used to set the orientation of the screen to vertical or landscape |
| 4 | **EXTRA_FULL_SCREEN**<br>It is used to control the user interface of the ViewImage |
| 5 | **INTENT_ACTION_VIDEO_CAMERA**<br>This intent is used to launch the camera in the video mode |
| 6 | **EXTRA_SIZE_LIMIT**<br>It is used to specify the size limit of video or image capture size |

Now you will use the function *startActivityForResult()* to launch this activity and wait for its result. Its syntax is given below

```
startActivityForResult(intent,0)
```

This method has been defined in the **activity** class. We are calling it from main activity. There are methods defined in the activity class that does the same job , but used when you are not calling from the activity but from somewhere else. They are listed below

| Sr.No | Activity function description |
|---|---|
| 1 | **startActivityForResult(Intent intent, int requestCode, Bundle options)**<br>It starts an activity , but can take extra bundle of options with it |
| 2 | **startActivityFromChild(Activity child, Intent intent, int requestCode)**<br>It launch the activity when your activity is child of any other activity |
| 3 | **startActivityFromChild(Activity child, Intent intent, int requestCode, Bundle options)**<br>It work same as above , but it can take extra values in the shape of bundle with it |
| 4 | **startActivityFromFragment(Fragment fragment, Intent intent, int requestCode)**<br>It launches activity from the fragment you are currently inside |
| 5 | **startActivityFromFragment(Fragment fragment, Intent intent, int requestCode, Bundle options)**<br>It not only launches the activity from the fragment , but can take extra values with it |

No matter which function you used to launch the activity , they all return the result. The result can be obtained by overriding the function *onActivityResult*.

## Android - Wi-Fi

Android allows applications to access to view the access the state of the wireless connections at very low level. Application can access almost all the information of a wifi connection.

The information that an application can access includes connected network's link speed,IP address, negotiation state, other networks information. Applications can also scan, add, save, terminate and initiate Wi-Fi connections.

Android provides **WifiManager** API to manage all aspects of WIFI connectivity. We can instantiate this class by calling **getSystemService** method. Its syntax is given below −

```
WifiManager mainWifiObj;
mainWifiObj = (WifiManager) getSystemService(Context.WIFI_SERVICE);
```

In order to scan a list of wireless networks, you also need to register your BroadcastReceiver. It can be registered using **registerReceiver** method with argument of your receiver class object. Its syntax is given below −

```
class WifiScanReceiver extends BroadcastReceiver {
  public void onReceive(Context c, Intent intent) {
  }
}

WifiScanReceiver wifiReciever = new WifiScanReceiver();
registerReceiver(wifiReciever, new
IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
```

The wifi scan can be start by calling the **startScan** method of the WifiManager class. This method returns a list of ScanResult objects. You can access any object by calling the **get** method of list. Its syntax is given below −

```
List<ScanResult> wifiScanList = mainWifiObj.getScanResults();
String data = wifiScanList.get(0).toString();
```

Apart from just Scanning, you can have more control over your WIFI by using the methods defined in WifiManager class. They are listed as follows −

| Sr.No | Method & Description |
|---|---|
| 1 | **addNetwork(WifiConfiguration config)**<br>This method add a new network description to the set of configured networks. |
| 2 | **createWifiLock(String tag)**<br>This method creates a new WifiLock. |
| 3 | **disconnect()**<br>This method disassociate from the currently active access point. |
| 4 | **enableNetwork(int netId, boolean disableOthers)**<br>This method allow a previously configured network to be associated with. |

| | |
|---|---|
| 5 | **getWifiState()**<br>This method gets the Wi-Fi enabled state |
| 6 | **isWifiEnabled()**<br>This method return whether Wi-Fi is enabled or disabled. |
| 7 | **setWifiEnabled(boolean enabled)**<br>This method enable or disable Wi-Fi. |
| 8 | **updateNetwork(WifiConfiguration config)**<br>This method update the network description of an existing configured network. |

## Android - Widgets

A widget is a small gadget or control of your android application placed on the home screen. Widgets can be very handy as they allow you to put your favourite applications on your home screen in order to quickly access them. You have probably seen some common widgets, such as music widget, weather widget, clock widget e.t.c

Widgets could be of many types such as information widgets, collection widgets, control widgets and hybrid widgets. Android provides us a complete framework to develop our own widgets.

### Widget - XML file

In order to create an application widget, first thing you need is AppWidgetProviderInfo object, which you will define in a separate widget XML file. In order to do that, right click on your project and create a new folder called **xml**. Now right click on the newly created folder and create a new XML file. The resource type of the XML file should be set to **AppWidgetProvider**. In the xml file, define some properties which are as follows −

```
<appwidget-provider
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:minWidth="146dp"
  android:updatePeriodMillis="0"
  android:minHeight="146dp"
  android:initialLayout="@layout/activity_main">
</appwidget-provider>
```

**Widget - Layout file**

Now you have to define the layout of your widget in your default XML file. You can drag components to generate auto xml.

**Widget - Java file**

After defining layout, now create a new JAVA file or use existing one, and extend it with **AppWidgetProvider** class and override its update method as follows.

In the update method, you have to define the object of two classes which are PendingIntent and RemoteViews. Its syntax is −

```
PendingIntent pending = PendingIntent.getActivity(context, 0, intent, 0);
RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.activity_main);
```

In the end you have to call an update method updateAppWidget() of the AppWidgetManager class. Its syntax is −

appWidgetManager.updateAppWidget(currentWidgetId,views);

A part from the updateAppWidget method, there are other methods defined in this class to manipulate widgets. They are as follows −

| Sr.No | Method & Description |
|---|---|
| 1 | **onDeleted(Context context, int[] appWidgetIds)**<br>This is called when an instance of AppWidgetProvider is deleted. |
| 2 | **onDisabled(Context context)**<br>This is called when the last instance of AppWidgetProvider is deleted |
| 3 | **onEnabled(Context context)**<br>This is called when an instance of AppWidgetProvider is created. |
| 4 | **onReceive(Context context, Intent intent)**<br>It is used to dispatch calls to the various methods of the class |

**Widget - Manifest file**

You also have to declare the AppWidgetProvider class in your manifest file as follows:

```
<receiver android:name="ExampleAppWidgetProvider" >

  <intent-filter>
    <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
  </intent-filter>

  <meta-data android:name="android.appwidget.provider"
    android:resource="@xml/example_appwidget_info" />
</receiver>
```

## Android - XML Parser

XML stands for Extensible Mark-up Language.XML is a very popular format and commonly used for sharing data on the internet. This chapter explains how to parse the XML file and extract necessary information from it.

Android provides three types of XML parsers which are **DOM,SAX and XMLPullParser**. Among all of them android recommend XMLPullParser because it is efficient and easy to use. So we are going to use XMLPullParser for parsing XML.

The first step is to identify the fields in the XML data in which you are interested in. For example. In the XML given below we interested in getting temperature only.

```
<?xml version="1.0"?>
<current>

  <city id="2643743" name="London">
    <coord lon="-0.12574" lat="51.50853"/>
    <country>GB</country>
    <sun rise="2013-10-08T06:13:56" set="2013-10-08T17:21:45"/>
  </city>

  <temperature value="289.54" min="289.15" max="290.15" unit="kelvin"/>
  <humidity value="77" unit="%"/>
  <pressure value="1025" unit="hPa"/>
</current>
```

**XML - Elements**

An xml file consist of many components. Here is the table defining the components of an XML file and their description.

| Sr.No | Component & description |
|---|---|
| 1 | **Prolog**<br>An XML file starts with a prolog. The first line that contains the information about a file is prolog |
| 2 | **Events**<br>An XML file has many events. Event could be like this. Document starts , Document ends, Tag start , Tag end and Text e.t.c |
| 3 | **Text**<br>Apart from tags and events, and xml file also contains simple text. Such as **GB** is a text in the country tag. |
| 4 | **Attributes**<br>Attributes are the additional properties of a tag such as value e.t.c |

**XML - Parsing**

In the next step, we will create XMLPullParser object , but in order to create that we will first create XmlPullParserFactory object and then call its newPullParser() method to create XMLPullParser. Its syntax is given below −

```
private XmlPullParserFactory xmlFactoryObject = XmlPullParserFactory.newInstance();
private XmlPullParser myparser = xmlFactoryObject.newPullParser();
```

The next step involves specifying the file for XmlPullParser that contains XML. It could be a file or could be a Stream. In our case it is a stream.Its syntax is given below −

```
myparser.setInput(stream, null);
```

The last step is to parse the XML. An XML file consist of events, Name, Text, AttributesValue e.t.c. So XMLPullParser has a separate function for parsing each of the component of XML file. Its syntax is given below −

```
int event = myParser.getEventType();
while (event != XmlPullParser.END_DOCUMENT)  {
  String name=myParser.getName();
  switch (event){
    case XmlPullParser.START_TAG:
    break;

    case XmlPullParser.END_TAG:
    if(name.equals("temperature")){
      temperature = myParser.getAttributeValue(null,"value");
    }
    break;
  }
  event = myParser.next();
}
```

The method **getEventType** returns the type of event that happens. e.g: Document start , tag start e.t.c. The method **getName** returns the name of the tag and since we are only interested in temperature , so we just check in conditional statement that if we got a temperature tag , we call the method **getAttributeValue** to return us the value of temperature tag.

Apart from the these methods, there are other methods provided by this class for better parsing XML files. These methods are listed below –

| Sl.No | Method & description |
|---|---|
| 1 | **getAttributeCount()**: This method just Returns the number of attributes of the current start tag |
| 2 | **getAttributeName(int index)**: This method returns the name of the attribute specified by the index value |
| 3 | **getColumnNumber()**: This method returns the Returns the current column number, starting from 0. |

| | |
|---|---|
| 4 | **getDepth()**: This method returns Returns the current depth of the element. |
| 5 | **getLineNumber()**: Returns the current line number, starting from 1. |
| 6 | **getNamespace()**: This method returns the name space URI of the current element. |
| 7 | **getPrefix()**: This method returns the prefix of the current element |
| 8 | **getName()**: This method returns the name of the tag |
| 9 | **getText()**: This method returns the text for that particular element |
| 10 | **isWhitespace()**: This method checks whether the current TEXT event contains only whitespace characters. |