

Android - Sending SMS

In Android, you can use SmsManager API or devices Built-in SMS application to send SMS's. In this tutorial, we shows you 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);
```

Of course, 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) This method divides a message text into several fragments, none bigger than the maximum SMS message size.
2	static SmsManager getDefault() 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) 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) Send a multi-part text based SMS.

5	void sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent) Send a text based SMS.
---	---

Example

Following example shows you in practical how to use SmsManager object to send an SMS to the given mobile number.

To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

Step	Description
1	You will use Android Studio IDE to create an Android application and name it as <i>tutorialspoint</i> under a package <i>com.example.tutorialspoint</i> .
2	Modify <i>src/MainActivity.java</i> file and add required code to take care of sending sms.
3	Modify layout XML file <i>res/layout/activity_main.xml</i> add any GUI component if required. I'm adding a simple GUI to take mobile number and SMS text to be sent and a simple button to send SMS.
4	No need to define default string constants at <i>res/values/strings.xml</i> . Android studio takes care of default constants.
5	Modify <i>AndroidManifest.xml</i> as shown below
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.tutorialspoint/MainActivity.java**.

```
package com.example.tutorialspoint;

import android.Manifest;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.app.Activity;

import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.telephony.SmsManager;

import android.util.Log;
import android.view.Menu;
```

```

import android.view.View;

import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {
    private static final int MY_PERMISSIONS_REQUEST_SEND_SMS = 0;
    Button sendBtn;
    EditText txtphoneNo;
    EditText txtMessage;
    String phoneNo;
    String message;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        sendBtn = (Button) findViewById(R.id.btnSendSMS);
        txtphoneNo = (EditText) findViewById(R.id.editText);
        txtMessage = (EditText) findViewById(R.id.editText2);

        sendBtn.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                sendSMSMessage();
            }
        });
    }

    protected void sendSMSMessage() {
        phoneNo = txtphoneNo.getText().toString();
        message = txtMessage.getText().toString();

        if (ContextCompat.checkSelfPermission(this,
            Manifest.permission.SEND_SMS)
            != PackageManager.PERMISSION_GRANTED) {
            if (ActivityCompat.shouldShowRequestPermissionRationale(this,
                Manifest.permission.SEND_SMS)) {
            } else {
                ActivityCompat.requestPermissions(this,
                    new String[]{Manifest.permission.SEND_SMS},
                    MY_PERMISSIONS_REQUEST_SEND_SMS);
            }
        }
    }

    @Override

```

```

public void onRequestPermissionsResult(int requestCode,String permissions[], int[]
grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_SEND_SMS: {
            if (grantResults.length > 0
                && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                SmsManager smsManager = SmsManager.getDefault();
                smsManager.sendTextMessage(phoneNo, null, message, null, null);
                Toast.makeText(getApplicationContext(), "SMS sent.",
                    Toast.LENGTH_LONG).show();
            } else {
                Toast.makeText(getApplicationContext(),
                    "SMS failed, please try again.", Toast.LENGTH_LONG).show();
                return;
            }
        }
    }
}

```

Following will be the content of **res/layout/activity_main.xml** file –

Here abc indicates about tutorialspoint logo

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="MainActivity">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Sending SMS Example"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:textSize="30dp" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Tutorials point "

```

```

    android:textColor="#ff87ff09"
    android:textSize="30dp"
    android:layout_below="@+id/textView1"
    android:layout_alignRight="@+id/imageButton"
    android:layout_alignEnd="@+id/imageButton" />

```

```

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_below="@+id/textView2"
    android:layout_centerHorizontal="true" />

```

```

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:hint="Enter Phone Number"
    android:phoneNumber="true"
    android:textColorHint="@color/abc_primary_text_material_dark"
    android:layout_below="@+id/imageButton"
    android:layout_centerHorizontal="true" />

```

```

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText2"
    android:layout_below="@+id/editText"
    android:layout_alignLeft="@+id/editText"
    android:layout_alignStart="@+id/editText"
    android:textColorHint="@color/abc_primary_text_material_dark"
    android:layout_alignRight="@+id/imageButton"
    android:layout_alignEnd="@+id/imageButton"
    android:hint="Enter SMS" />

```

```

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Send Sms"
    android:id="@+id/btnSendSMS"
    android:layout_below="@+id/editText2"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="48dp" />

```

```
</RelativeLayout>
```

Following will be the content of **res/values/strings.xml** to define two new constants –

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

```
<resources>
  <string name="app_name">tutorialspoint</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** –

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

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

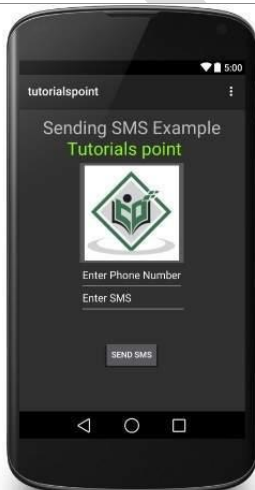
  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name="com.example.tutorialspoint.MainActivity"
      android:label="@string/app_name" >

      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>

    </activity>

  </application>
</manifest>
```



Now you can enter a desired mobile number and a text message to be sent on that number. Finally click on **Send SMS** button to send your SMS. Make sure your GSM/CDMA connection is working fine to deliver your SMS to its recipient.

You can take a number of SMS separated by comma and then inside your program you will have to parse them into an array string and finally you can use a loop to send message to all the given numbers. That's how you can write your own SMS client. Next section will show you how to use existing SMS client to send SMS.

Using Built-in Intent to send SMS

You can use Android Intent to send SMS by calling built-in SMS functionality of the Android. Following section explains different parts of our Intent object required to send an SMS.

Intent Object - Action to send SMS

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

```
Intent smsIntent = new Intent(Intent.ACTION_VIEW);
```

Intent Object - Data/Type to send SMS

To send an SMS you need to specify **smsto:** as URI using setData() method and data type will be to **vnd.android-dir/mms-sms** using setType() method as follows –

```
smsIntent.setData(Uri.parse("smsto:"));  
smsIntent.setType("vnd.android-dir/mms-sms");
```

Intent Object - Extra to send SMS

Android has built-in support to add phone number and text message to send an SMS as follows –

```
smsIntent.putExtra("address" , new String("0123456789;3393993300"));  
smsIntent.putExtra("sms_body" , "Test SMS to Angilla");
```

Here address and sms_body are case sensitive and should be specified in small characters only. You can specify more than one number in single string but separated by semi-colon (;).

Example

Following example shows you in practical how to use Intent object to launch SMS client to send an SMS to the given recipients.

To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as <i>tutorialspoint</i> under a package <i>com.example.tutorialspoint</i> .

2	Modify <i>src/MainActivity.java</i> file and add required code to take care of sending SMS.
3	Modify layout XML file <i>res/layout/activity_main.xml</i> add any GUI component if required. I'm adding a simple button to launch SMS Client.
4	No need to define default constants.Android studio takes care of default constants.
5	Modify <i>AndroidManifest.xml</i> as shown below
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.tutorialspoint/MainActivity.java**.

```
package com.example.tutorialspoint;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button startBtn = (Button) findViewById(R.id.button);
        startBtn.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                sendSMS();
            }
        });
    }

    protected void sendSMS() {
        Log.i("Send SMS", "");
        Intent smsIntent = new Intent(Intent.ACTION_VIEW);

        smsIntent.setData(Uri.parse("smsto:"));
        smsIntent.setType("vnd.android-dir/mms-sms");
        smsIntent.putExtra("address" , new String ("01234"));
        smsIntent.putExtra("sms_body" , "Test ");
    }
}
```



```

try {
    startActivity(smsIntent);
    finish();
    Log.i("Finished sending SMS...", "");
} catch (android.content.ActivityNotFoundException ex) {
    Toast.makeText(MainActivity.this,
        "SMS failed, please try again later.", Toast.LENGTH_SHORT).show();
}
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}

```

Following will be the content of **res/layout/activity_main.xml** file –

Here abc indicates about tutorialspoint logo

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Drag and Drop Example"
        android:id="@+id/textView"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:textSize="30dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Tutorials Point "
        android:id="@+id/textView2"
        android:layout_below="@+id/textView"
        android:layout_centerHorizontal="true"

```

```
        android:textSize="30dp"
        android:textColor="#ff14be3c" />

<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageView"
    android:src="@drawable/abc"
    android:layout_marginTop="48dp"
    android:layout_below="@+id/textView2"
    android:layout_centerHorizontal="true" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Compose SMS"
    android:id="@+id/button"
    android:layout_below="@+id/imageView"
    android:layout_alignRight="@+id/textView2"
    android:layout_alignEnd="@+id/textView2"
    android:layout_marginTop="54dp"
    android:layout_alignLeft="@+id/imageView"
    android:layout_alignStart="@+id/imageView" />
```

</RelativeLayout>

Following will be the content of **res/values/strings.xml** to define two new constants –

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">tutorialspoint</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** –

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

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="com.example.tutorialspoint.MainActivity"
            android:label="@string/app_name" >

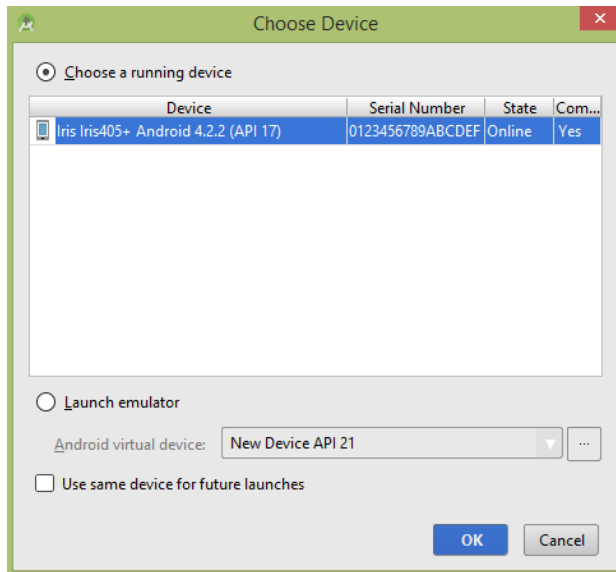
            <intent-filter>
```

Module 3

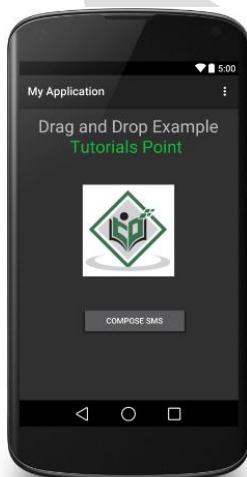
```
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>

</activity>

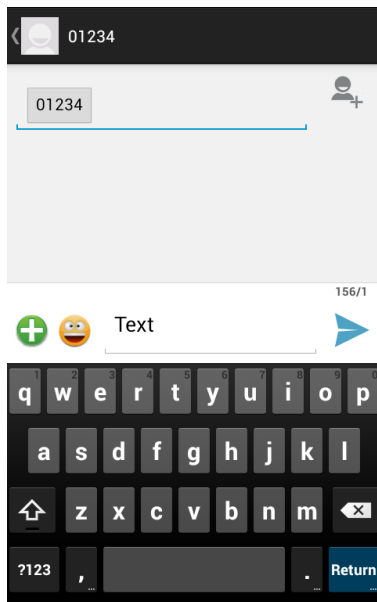
</application>
</manifest>
```



Select your mobile device as an option and then check your mobile device which will display following screen –



Now use **Compose SMS** button to launch Android built-in SMS clients which is shown below –



You can modify either of the given default fields and finally use send SMS button to send your SMS to the mentioned recipient.

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.

This chapter lists down all the simple steps to create an application which can be used to make a Phone Call. You 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.

Example

Following example shows you in practical how to use Android Intent to make phone call to the given mobile number.

To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as <i>My Application</i> under a package <i>com.example.saira_000.myapplication</i> .
2	Modify <i>src/MainActivity.java</i> file and add required code to take care of making a call.
3	Modify layout XML file <i>res/layout/activity_main.xml</i> add any GUI component if required. I'm adding a simple button to Call 91-000-000-0000 number
4	No need to define default string constants.Android studio takes care of default constants.
5	Modify <i>AndroidManifest.xml</i> as shown below
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/MainActivity.java**.

```
package com.example.saira_000.myapplication;

import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.net.Uri;
import android.os.Bundle;
import android.support.v4.app.ActivityCompat;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {
    private Button button;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```

super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
button = (Button) findViewById(R.id.buttonCall);

button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View arg0) {
        Intent callIntent = new Intent(Intent.ACTION_CALL);
        callIntent.setData(Uri.parse("tel:0377778888"));

        if (ActivityCompat.checkSelfPermission(MainActivity.this,
            Manifest.permission.CALL_PHONE) != PackageManager.PERMISSION_GRANTED) {
            return;
        }
        startActivity(callIntent);
    }
});
}
}

```

Following will be the content of **res/layout/activity_main.xml** file –

```

<?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" >

    <Button
        android:id="@+id/buttonCall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="call 0377778888" />

</LinearLayout>

```

Following will be the content of **res/values/strings.xml** to define two new constants –

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My Application</string>
</resources>

```

Following is the default content of **AndroidManifest.xml** –

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.saira_000.myapplication" >

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

```

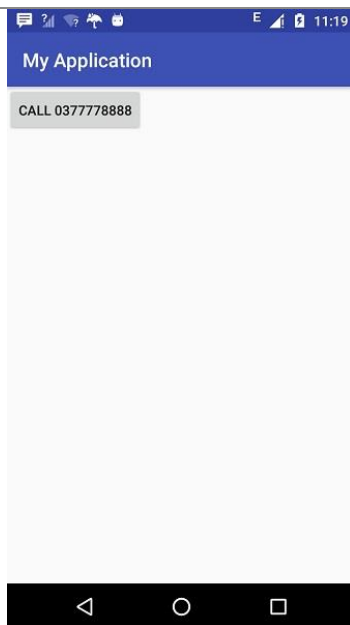
```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
        android:name="com.example.saira_000.myapplication.MainActivity"
        android:label="@string/app_name" >

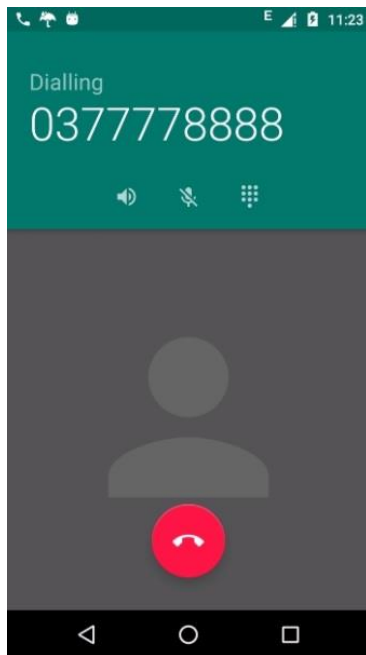
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>

    </activity>

</application>
</manifest>
```

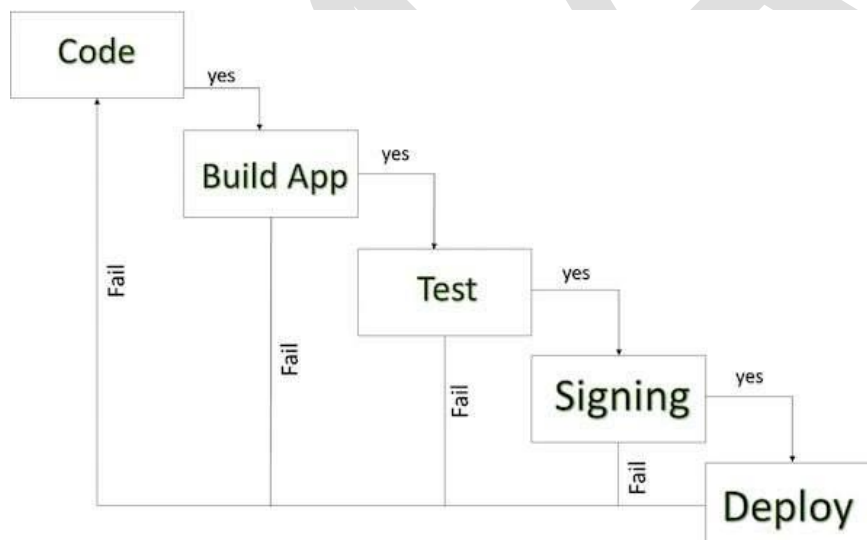


Now use **Call** button to make phone call as shown below –



Publishing Android Application

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



Android development life cycle

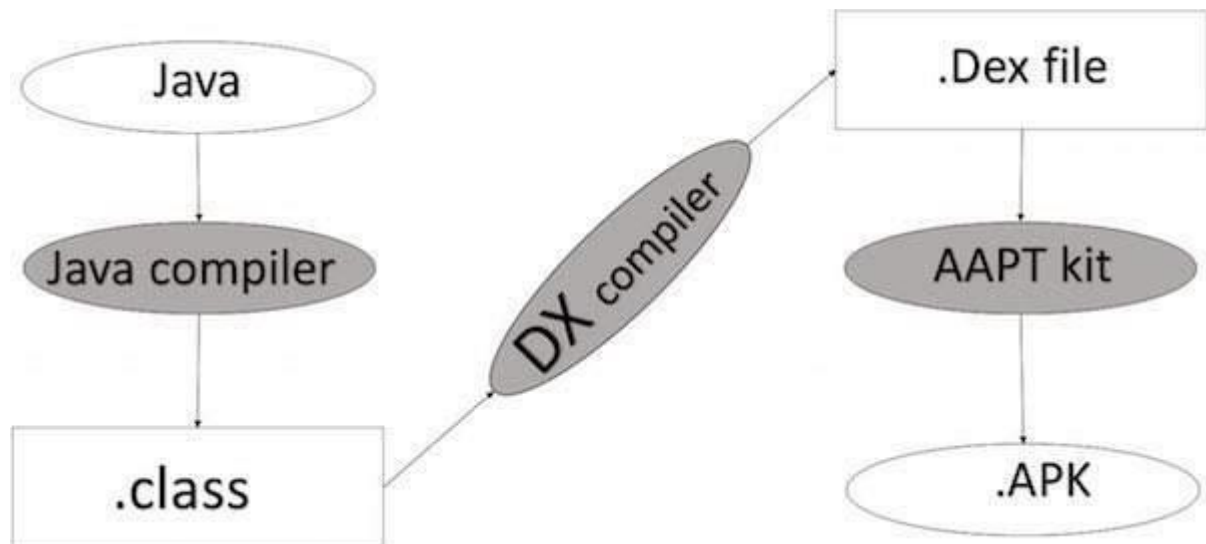
Once you developed and fully tested your Android Application, you can start selling or distributing free using Google Play (A famous Android marketplace). You can also release your applications by sending them directly to users or by letting users download them from your own website.

You can check a detailed publishing process at Android official website, but this tutorial will take you through simple steps to launch your application on Google Play. 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

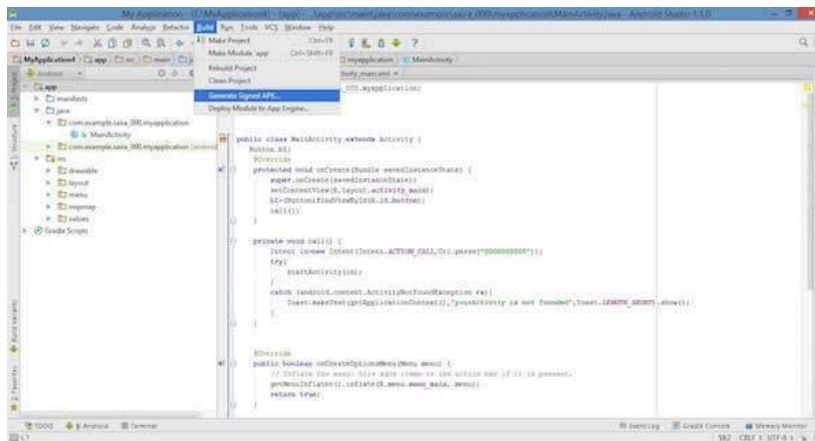
Before exporting the apps, you must use some of the tools

- **Dx tools**(Dalvik executable tools): It goes 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.

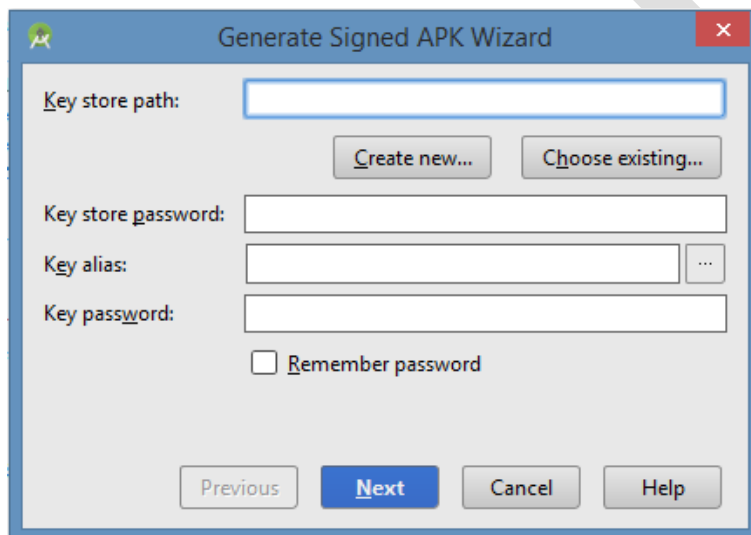
You will need to export your application as an APK (Android Package) file before you upload it to Google Play marketplace.

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 –

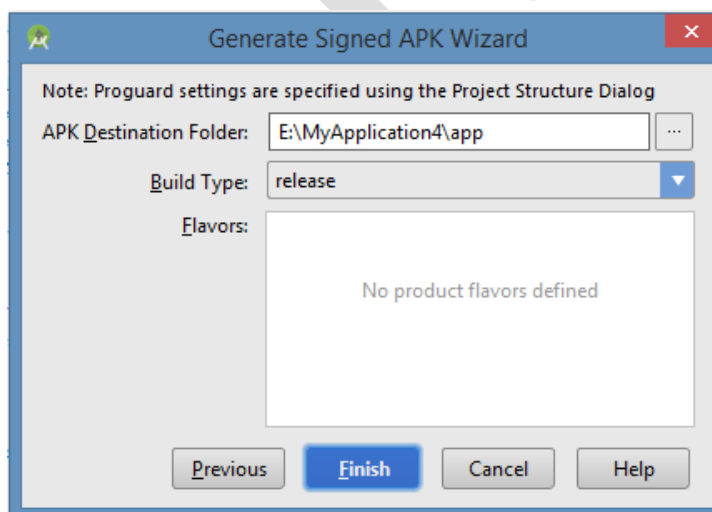
Module 3



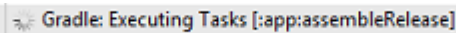
Next select, **Generate Signed APK** option as shown in the above screen shot and then click it so that you get following screen where you will choose **Create new keystore** to store your application.



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 –



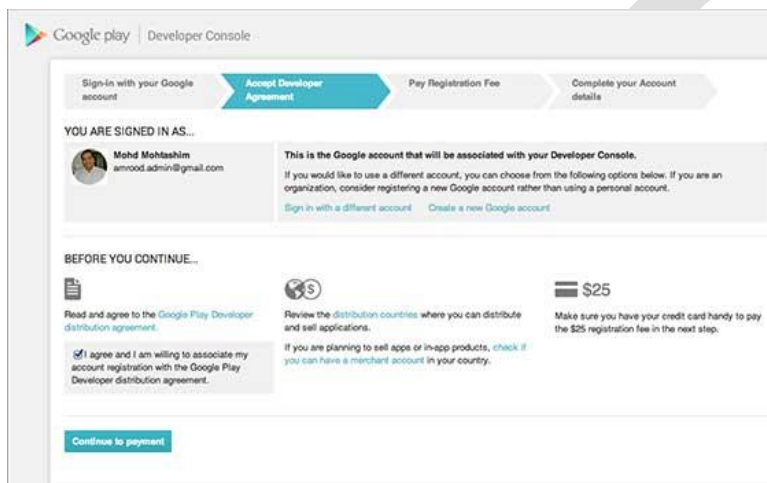
Once you filled up all the information, like app destination, build type and flavours click **finish** button. While creating an application it will show as below



Finally, it will generate your Android Application as APK format 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
```

Module 3

- 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
```

MAP